68' MICRO JOURNAL

Australia	A $ 4.75	New Zealand	NZ $ 6.50
Singapore	S $ 9.45	Hong Kong	H $23.50
Malaysia	M $ 9.45	Sweden	30-SEK

$2.95 USA

* Motorola 68020 *
6809-68008-68000-68010

VOLUME VIII ISSUE VI • Devoted to the 68XX User • June 1986

"Small Computers Doing Big Things"

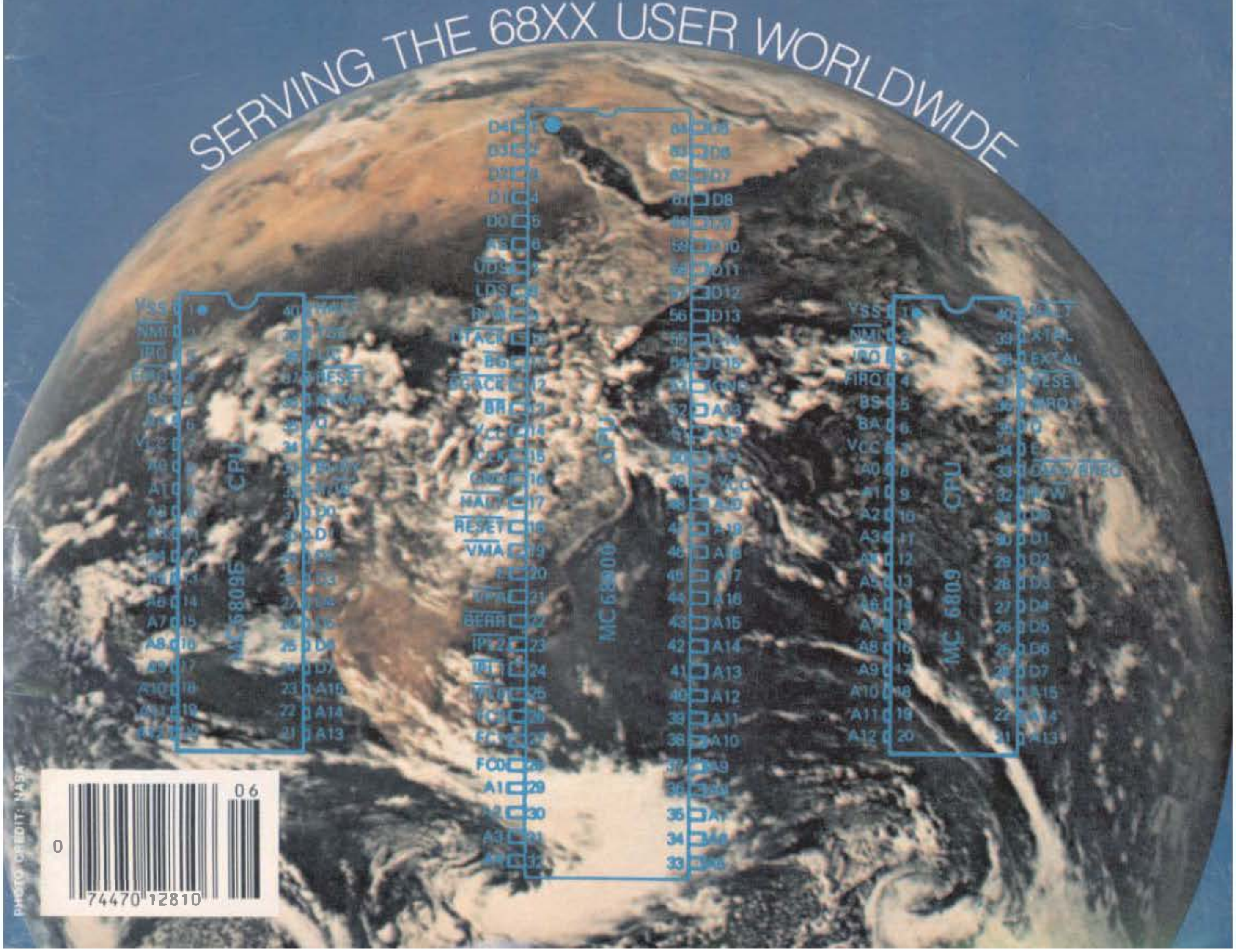


PHOTO CREDIT: NASA

0 74470 12810 06

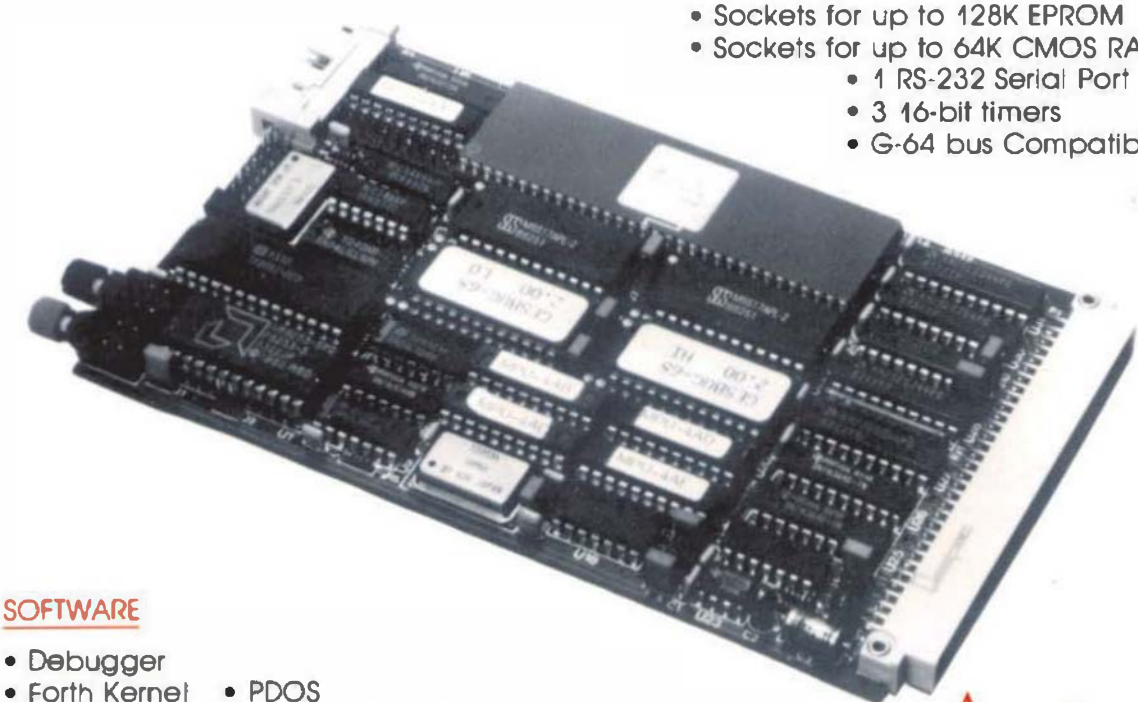
68000 CPU FOR $395

QUANTITY ONE

GESMPU-4A

FEATURES

- 8 MHZ, 16/32-bit 68000 CPU
- Sockets for up to 128K EPROM
- Sockets for up to 64K CMOS RAM
- 1 RS-232 Serial Port
- 3 16-bit timers
- G-64 bus Compatible



SOFTWARE

- Debugger
- Forth Kernel
- CP/M 68K
- OS-9
- PDOS
- C-Basic-Pascal
- Editor-Assembler

OS-9 AVAILABLE

Call for your free data sheet and
our 100 page catalog of board level products.
1-800-443-7722

Gespac

USA — CANADA
100 West Hoover Ave.-11
Mesa, AZ 85202
Tel. (602) 962-5559
Telex 386 575

INTERNATIONAL
3,chemin des Aulx
CH-1228 Geneva
Tel. (022) 713 400
Telex 429 989

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.

HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state Instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

UNIX is a trademark of A.T. & T.
ADA is a trademark of the U.S. Government.
UniFLEX is a trademark of Technical Systems Consultants, Inc.
GMX and GIMIX are trademarks of GIMIX, Inc.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS - HARDWARE

Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S,9 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS - WORD PROCESSORS

Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Editorial Staff

Don Williams Sr. .... Publisher
Larry E. Williams .... Executive Editor
Tom E. Williams .... Production Editor
Robert L. Nay .... Technical Editor

Administrative Staff

Mary Robertson .... Officer Manager
Joyce Williams .... Subscriptions
Christine Lea .... Accounting

Contributing Editors

Ron Anderson	Norm Commo
Peter Dibble	William E. Fisher
Dr. Theo Elbert	Carl Mann
Dr. E.M. Pass	Ron Voigts
Philip Lucido	Randy Lewis

Special Technical Projects

Clay Abrams K6AEP
Tom Hunt

CONTENTS

Vol. VIII, Issue 6 June 1986

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

Copyrighted 1985 by Computer Publishing Inc.

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add **$9.50** a Year. Other **Foreign Add $12 a Year for Surface, Airmail Add $48 a Year.** Must be in U.S. **currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be **on either 5 or 8 inch disk** in STYLOGRAPH or TSC Editor format with **3.5 inch column width**. All disks will be returned. Articles submitted **on paper** should be **4.5 inches in width** (Including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: **.sp** space, **.pp** paragraph, **.fl** fill and **.nf** no fill. Also please **do not format** within the text with **multiple spaces**. We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except **.pg** page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **must be signed**. Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance**. No classified ads accepted over the phone.

The VME BUS and OS-9:

Ultimate Software for the Ultimate Bus.

Modularity. Flexibility. High Performance. Future growth. These are probably the prime reasons you chose the VME bus. Why not use the same criteria when selecting your system software? That's why you should take a look at Microware's OS-9/68000 Operating System—it's the perfect match for the VME bus.

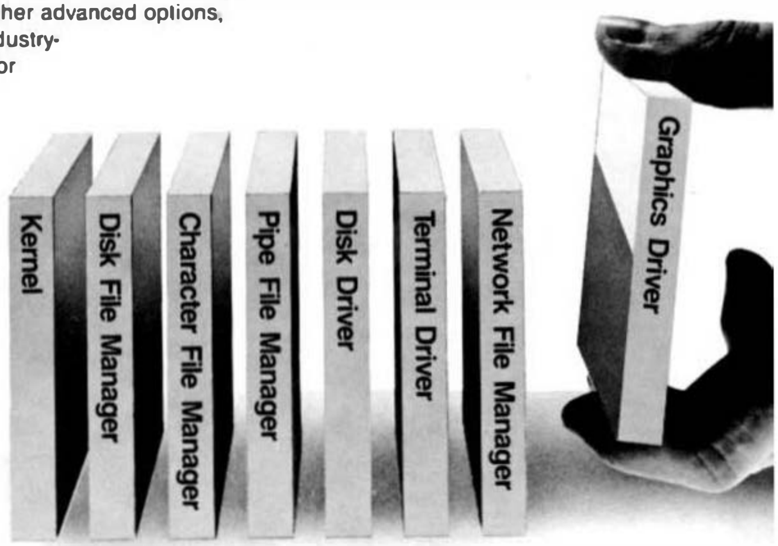
When you're working with VME you must have access to every part of the system. Unlike other operating systems that literally scream KEEP OUT!, OS-9's open architecture invites you to create, adapt, customize and expand. Thanks to its unique modular design, OS-9 naturally fits virtually any system, from simple ROM-based controllers up to large multiuser systems.

And that's just the beginning of the story. OS-9 gives you a complete UNIX-application compatible environment. It is multitasking, real time, and extremely fast. And if you're still not impressed, consider that a complete OS-9 executive and I/O driver package typically fits in less than 24K of RAM or ROM.

Software tools abound for OS-9, including outstanding Microware C, Basic, Fortran, and Pascal compilers. In addition, cross C compilers and cross assemblers are available for VAX systems under Unix or VMS. You can also plug in other advanced options, such as the GSS-DRIVERS™ Virtual Device Interface for industry-standard graphics support, or the OS-9 Network File Manager for high level, hardware-independent networking.

Designed for the most demanding OEM requirements, OS-9's performance and reliability has been proven in an incredible variety of applications. There's nothing like a track record as proof: to date, over 200 OEMs have shipped more than 100,000 OS-9-based systems.

Ask your VME system supplier about OS-9. Or you can install and evaluate OS-9 on your own custom system with a reasonably priced Microware PortPak™. Contact Microware today. We'll send you complete information about OS-9 and a list of quality manufacturers who offer off-the-shelf VME/OS-9 packages.



Modular Hardware Deserves Modular Software

MICROWARE.

Microware Systems Corporation
1866 N.W. 114th Street • Des Moines, Iowa 50322
Phone 515-224-1929 • Telex 910-520-2535

Microware Japan, Ltd.
41-19 Honcho 4-Chome, Funabashi City • Chiba 273,
Japan • Phone 0473 (28) 4493 • Telex 781-299-3122

Micromaster Scandinavian AB
S:t Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
Phone: 018-138595
Telex: 76129

Dr. Rudolf Keil, GmbH
Porphystrasse 15
D-6905 Schriesheim
West Germany
Phone: (0 62 03) 67 41
Telex: 465025

Elsoft AG
Zelweg 12
CH-5405 Baden-Dattwil
Switzerland
Phone: (056) 83-3377
Telex: 828275

Vivaway Ltd.
36-38 John Street
Luton, Bedfordshire, LU1 2JE
United Kingdom
Phone: (0582) 423425
Telex: 825115

Microprocessor Consultants
92 Bynya Road
Palm Beach 2108
NSW Australia
Phone: 02-919-4917

Microdata Soft
97 bis, rue de Colombes
92400 Courbevoie
France
Phone: 1-768-80-80
Telex: 615405

OS-9 is a trademark of Microware and Motorola. PortPak is a trademark of Microware. GSS-Drivers is a trademark of Graphic Software Systems, Inc. VAX and VMS are trademarks of DEC. Unix is a trademark of AT&T.

MUSTANG -020 Super SBC ™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the REAL KICK!"
Only from DATA-COMP

This is the NCC, world beater GMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your specs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

This system is the state-of-the-art star-ship. It runs rings around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be Turtle or Thoroughbred?

MUSTANG-020 System Prices
12.5 Mhz

Mustang-020 SBC, wired & tested with 4 DB25 Serial ports pre-wired, ready to install with your cabinet, P/S, CRT and drives.....$2750.00

M020 Cabinet and P/S, for Mustang-020, less cables............................... $299.95

M020 Cables, dual floppy or HD specify which floppy or winchester....................$39.95

M020FC Floppy cabinet and P/S, holds and powers 2 thin-line floppies...................$79.95

M020F Floppy, 80 track, DD/DS..........$269.95

OS-9, SPECIAL Mustang-020 version......$350.00

MC6881 F/P co-processor................$495.00

20 Megabyte Winchester.................$895.00

Winchester HD Cootroller.XEBEC.........$395.00

8 Port Expansion (complete)............$495.00

** Special Winchester Notice **

The Mustang-020 device descriptors will allow you to use practically **ANY** winchester drive supported by XEBEC 1410/1410A or OMTI 20C-1 controllers.

. . .

Include: $3.50 SBC, cables only S/H. Cabinets include $7.50 S/H. Complete System include $20.00. All checks must be in USA funds. Overseas specify shipping instructions and sufficient funds.

Dealer & Quantity Discounts Available

* Mustang-020 is trademark of Data-Comp-CPI

DATA-COMP 5900 Cassandra Smith Rd. Hixson, TN 37343		SHIPPING USA ADD 2% FOREIGN ADD 5% MIN. $2.50	(615)842-4600 For Ordering TELEX 5106006630

MUSTANG-020 Benchmarks ** Time Seconds		
Type System	32 bit Int. Loop	Register Long Loop
IBM AT 7300 Xenix Sys 3	9.7	No Registers
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " " "	5.1	3.2
68008 OS9 68K 8 Mhz	18.0	9.0
68000 " " 10 Mhz	6.5	4.0
MUSTANG-020 68020 MC68881 OS9 16 Mhz	2.2	0.88
MUSTANG-020 68020 MC68881 UNIFLEX "	1.8	1.22

```
** Loop: Main()
         {
           register long i;
           for (i=0; i < 999999; ++i);
         }
```

Estimated MIPS - MUSTANG-020 - 2.5 MIPS
Motorola Specs: Burst up to 7 - 8 MIPS - 16 Mhz

For a limited time we will offer $400 Trade-In on your old Q--- 68008 or 68000 SBC, must be working properly and complete with all software and cables. Call for more information!

Mustang-020 Software

OS-9

OS-9.............$350.00
Basic09...........300.00
C Compiler........400.00
Fortran 77...... *400.00
Pascal Compiler...400.00
OMEGASOFT-PASCAL..900.00
Stylo-Graph.......495.00
Stylo-Spell.......195.00
Stylo-Merge.......175.00
PAT...............229.00
JUST...............79.95
PAT/JUST Combo....249.50
SCULPTOR+.......**995.00
COM...............125.00
** See discount below

UniFLEX

UniFLEX..........$450.00
Screen Editor.....150.00
Sort-Merge........200.00
BASIC/PreCompiler.300.00
C Compiler........350.00
COBOL.............750.00
CMODEM.w/source...100.00
TMODEM.w/source...100.00
X-TALK.see.adv.....99.95
Cross Assembler....50.00
Fortran 77........450.00
SCULPTOR+.......**995.00
** See discount below

* New items
Standard system shipped 12.5 MHz
Add for 16 Mhz..68020....$400.00
Add for 16 Mhz..68881....$400.00
8 Port expansion board use two total of 20 RS232 ports wired & Tested............each....$498.00

SCULPTOR+...we are USA distributors for SCULPTOR+. Call or write for site license or multiple discounts.
SCULPTOR discount with complete system - Special - $695.00. OS-9 or UniFLEX.

** Software Discounts
Call for software discounts from 10-70% for buyers of these system , from Data-Comp. Limited offer.
Call!

Mustang 020 Features

- 12.5 MHz (16.6 MHz optional) MC68020 full 32-Bit wide Processor
  - 32-bit wide non-multiplexed data & address buses
  - On-chip instruction cache
  - Object-code compatible with earlier M68000 family processors (68000/68008/68010)
  - Enhanced instruction set - Coprocessor interface
- Optional 68881 Floating point Coprocessor (12.5 MHz or 16.6 MHz)
  - Direct extension of 68020 instruction set
  - Full support of IEEE P754, draft 10.0
  - Transcendentals and other math functions
- 2 Megabytes of RAM (512K x 32-bit organization)
- Up to 256K bytes of EPROM (64K X 32-bits)
  - Uses four 2764, 27128, 27256, or 27512 EPROMs
- 4 Asynchronous serial I/O ports (2 x MC68681 DUART)
  - Software programmable baud rates to 19.2K
  - Standard RS-232 interface
  - Optional network interface on one port
- 8 RS-232 Expansion ports (max. 20)
- Buffered 8-bit Parallel I/O Port (1/2 MC68230)
  - Centronics-type parallel printer pinout
  - May also be used as parallel input port
- Expansion Connector for Additional I/O Devices
  - 16-bit data path
  - 256 byte address space
  - 2 interrupt inputs
  - Clock and Control Signals
- Time-of-Day Clock/Calendar w/battery backup
- Controller for up to Two 5 1/4" Floppy Disk Drives
  - Single or double sided
  - Single or double density
  - 48 or 96 tracks per inch (40/80 Track)
- Mounts Directly to a Standard 5 1/4" Disk Drive
- SASI Interface for Intelligent Hard Disk Controllers
- Programmable Periodic Interrupt Generator
  - For time-slicing and real-time applications
  - Interrupt rates from microseconds to seconds
  - Highly Accurate timebase (5 PPM)
- 5-bit sense switch, readable by the processor
- Hardware single-step capability

**** ACTION PROVEN ****

The MUSTANG-020 is already on the job! And winning aclaim in industry, commerce, business and several government agencies. The delivery times were close to schedule. We are hearing back nothing but praise (and more orders).

If you are considering the purchase of a Mustang-020, be advised that the price will increase in the second quarter of this year.

Experienced users are awed at the tremendous power and speed of the Mustang-020, from Data-Comp. Especially when compared with other 68XXX systems. Not only is it more practical then all the others, but it is much more cost efficient. Compare it to any other 68XXX and you will see why!

Dual 5" 80 trk. Floppy No Winchester		Winchester & 1 Floppy	
020 Board	$2750.00		$2750.00
Cabinet	299.95		299.95
5"-80 trk Floppy(2)	539.90	(1)	269.95
Floppy cable	39.95		39.95
OS-9 68K	350.00		350.00
		Winchester cable	39.95
Total System	$3979.80	Winchester controller	395.00
Less 5%	-198.99	25 Mbyte Winchester	895.00
	$3780.81	Total System	$5039.80
S/H UPS	20.00	Less 5%	-251.99
Total	$3800.81		$4787.81
		S/H UPS	20.00
		Total	$4807.81

NOTE: 68881 Co-Processor Add $495.00
UniFLEX $450.00
less $350.00 (OS-9) Add $100.00

Prices and Specifications subject to change.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating **Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your **OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO C1	File load program to offset memory — ASM PIC
MEMOVE C1	Memory move program — ASM PIC
DUMP C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT C3	Parallel (enhanced) printer driver — ASM
MODEM C2	TTL output to CRT and modem (or other port) — ASM
SCPKG C1	Scientific math routines — PASCAL
U C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT C4	Parallel printer driver, without PFLAG — ASM
SET C5	Set printer modes — ASM
SETBAS1 C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
• All Currency in U.S. Dollars

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

Basically OS-9

Ron Voigts
2024 Baldwin Court
Glendale Heights, IL 60139

BEHIND THE SCENES

I've noticed that many times when a really good movie is at the theaters, there is usually a short release showing how the film was made. You can catch it on a cable station or the PBS network. The film short shows how the special effects were created, how the actors contributed to make the motion picture, and generally what went on behind the scenes. You would think seeing what went on behind scenes would make the motion picture less interesting. But I find (I think most people do) that seeing how it was done, improves my enjoyment of the film. The same can be said of the computer. You can enjoy it as it is, but if you know what goes on BEHIND THE SCENES, you'll enjoy it so much more.

In the April column I talked about the executable memory module. These are the modules that are found in the commands directory, usually CMDS. What makes them different from other files, is not what is in the disk descriptor, but what is in the file. The command module contains three parts--the header, the body and the CRC. We talked a few months ago about the body and CRC of the command module. I also mentioned the the execution offset address and the storage size for the module. These are extensions of the module header. They may not appear in the same in all modules. MOre about that later.

The first 8 bytes of the module are the header. They give important information about the module and its use. Many systems put this information on the disk, while the program itself tells nothing about its purpose. OS-9 differs from them. The disk file descriptor tells nothing about the modules purpose. The information about the executable module is located at the start of the module. Wherever the module goes, it carries along its own identification. The first 8 bytes are:

Address	Use
$00,$01	Sync bytes
$02,$03	Module Size
$04,$05	Module Name Offset
$06	Type and Language
$07	Attributes and Revision
$08	Header Parity Check

You could add in the Execution offset at $09,$0A and the Storage size at $0B,$0C. They are extension to the header. For now we'll concentrate on the first 8 bytes.

The **Sync Bytes** are the first two bytes of the module. They are always $87 and $CD. OS-9 uses them to recognize the start of module, hence their name. At start up, OS-9 looks for ROM modules by searching for the sync bytes. What makes them unique is that they don't correspond to any 6809 opcodes. Also, they don't occur in standard ASCII code. So the chances that they would occur except as the sync bytes are pretty remote.

The first 8 bytes of the module are the header. They give important information about the module and its use.... Wherever the module goes, it carries along its own identification.

Next is the **module size.** Not a whole lot that can be said about it. The two bytes here represent an integer value that tells the size of the module. The system can tell where module ends. And it can use the size to compute the CRC value for comparison to the value in the module.

The **Module Name Offset** is the number of bytes relative to the module start where the name begins. This is the name that gets placed in the module directory. It uses the system standard for string data. The last character of the name has its 8th bit set high. The modules edition number is placed after the name. It is a byte that indicates what version the module is. It is not necessary to have it there, but it is helpful if you're trying to distinguish between different version of the same module. Run IDENT on a module in the commands directory and it will tell the edition number.

Byte $06 is the **Type and Language Byte.** The 4 top bits are the module type. The 4 lower bits are the language indicator. A possible 15 codes can be used for the module type. However, I've only seen 7 used. They are:

Code	Purpose
$1	Program module
$2	Subroutine module
$4	Data module
$C	System module
$D	File manager
$E	Device Driver
$F	Device Descriptor

$3 was reserved for a Multi-module, but not yet used, $5-$B are user definable. And $0 is not used. For the language only 4 codes are used. They are:

Code	Purpose
$0	Data
$1	Object code
$2	I-code for Basic09
$3	P-code for Pascal

Originally, I-code for C Language, Cobol and Fortran were planned, but were never implemented. They would have been $4 through $6. Creating a type and language byte is simple. For example, if you created a device driver with your OS-9 assembler, its TYPE/LANG byte would be $E1. A Basic09 module would be $22.

The **Attribute and Revision Level** come next. This byte contains two pieces of information like the previous one. The top 4 bits tell whether the module is reentrant. If the top 4 bits equal $8 then the module is reentrant or 'sharable'. A sharable module can be used by a number of users. For example if you are running Basic09 and another user is running it, both of you are using the same module. Only you both have separate data areas. The revision level can range from $0 to $F. If two modules are loaded with the same name, the one with the higher revision level will be used. This can be reel handy if you want to replace a module that is in memory. OS-9 will always use the highest revision level.

Finally is the **Header Check**. The first 8 bytes are exclusively ORed together, otherwise know as XOR. To XOR two bytes, each bit is compared in the two bytes. If they match, the result is a 0 and if they differ the result is a 1. For example, let's XOR $83 and $D4. It is easiest to look at the bits.

```
$83   1 0 0 0 0 0 1 1
$D4   1 1 0 1 0 1 0 0
$57   0 1 0 1 0 1 1 1
```

Notice where that the 8th, 6th, and 4th bits are the same. They yield 0's. The other bits differ between the tow numbers, so the are 1. After XORing the first 8 bits the 1's compliment is taken. The result is subtracted from $FF. The final result is the header check. It is placed in the module header.

TEXT COMPRESSION

This month's programs deal with text compression. I wrote these two programs last year while trying to develop a method putting files on a disk with the least amount of space. With the advent of cheap memory and lower cost storage, test compression is not as important as it was at one time. But it can still be fun to play with it.

There are a number of methods to reduce text size. One method is to use a table of common words. Normally ASCII text uses only the first 128 bytes. These include bytes $00 thru $7F. The 8th bit never is set. Setting the 8th bit could be used to signal that the word should be looked up in the table. This would only account for 128 common words using byte $80 thru $FF. The processor could also extend the table and automatically use the next byte in line to access a table of 32768 words. That is quite a large table of common words. Most of the dictionaries I have seen for spell checkers aren't that large. The dictionaries size would have to be only perhaps 1000 words to cover most commonly used words. One letter words would not be in it. There is no sense in storing one byte words in two bytes. Two letter words would become a break even deal. Anything more would be a savings. This method is used. Many Basic languages tokenize keywords. A one or two byte token is used to represent a keyword in the language. The net result is reduced code size. The draw back to such a system is the overhead. Our table of common words would need a large storage area. Therefore, its savings in memory would not be realized until we started using texts or files that exceeded the table size and tokenized words.

Another method is to put more code in the allotted space. A byte is used to store one character. The code used is ASCII. It represents all printable and non-printable characters. Two bytes makes an integer which can hold two characters. What if we could put more in it? Imagine a character set shorter than the ASCII set. A very minimal set could be A thru Z, 0 thru 9, three punctuation and a carriage return character. This set comprises 40 characters in all. This method is called radix 40. Three characters are stored in the integer with the following formula:

$$((c1*40)+c2)*40)+c3$$

C1, c2 and c3 are now in the integer form. To remove them the process is reversed. The savings is 50% in text space. The disadvantages is a narrow character set. There are no lowercase letters and only a few punctuation are permitted. This would definitely limit what kind of text you could create.

The method I used in this months programs retains the full ASCII character set, while being easily implemented. Only 7 bits are used in the byte to store ASCII code. Normally the 8th bit is left unused. This location could be used to hide something. The most often occurring character in standard text is the space. If the space could be eliminated from the text file, there should be a considerable savings of space. The method used by PACK is to read a character from the standard input. If it is a space, set a flag. Then it reads another character. If the space flag is set, the 8th bit on the next character is set high and that character is written to the standard output path. Otherwise the character is written as it is read. The UNPACK program reads a packed file and restores it to its original form. If the character has its 8th bit set high, a space is put on the output path and then the character is written with the 8th bit cleared. If the 8th bit is not set, it is written as it was read.

Besides having fun with PACK and UNPACK, they give us a good example of how the module header is created. The MOD command in the source code is very important. In PACK, it reads:

```
     MOD     PACEND, NAME, $11, $81, START, PACMEM
```

It helps to create the module header. Here's how it works. The first two bytes of the header are $87 and $CD. The next two are the size, which is equal to PACEND. Next two are the offset to the name which is equal to NAME in the MOD list. Module type/Language type is $11, program module and object code. Attributes/Revision level is $81, sharable and revision 1. Check the tables I gave earlier for these two bytes. START is the execution offset and PACMEM is the permanent storage requirement. Once you've written and compiled PACK, use DUMP to examine the module header and see what is in the header. Also try analyzing the UNPACK module.

The MOD code can specify header information for other type of modules, too. Normally the last two arguments specify the entry point and data memory size. But they can vary with the module type. A subroutine module will have its last argument set to 0, since it will use the caller's memory. A device descriptor will use the last two positions in MOD to indicate where the file manager's name and driver's name are located in the module. The data module will indicate the data start and the data size. So, depending on the module, a number of different pieces of information can be conveyed with the module header extensions.

The best thing to do is to examine things for yourself. Use IDENT to look at the different modules. Also, DUMP the module to you printer and examine the first 8 bytes or so. It's fun to watch a good show. It is more fun to go BEHIND THE SCENES.

```
        ****************
        * PACK by Ron Voigts
        * 2-FEB-86
        *
        * This program will pack an ascii file
        * into a compact file by hiding
        * spaces.
        *
        * Usage: pack <infile >outfile
        *
                          nam   PACK
                          ttl   packs ascii files
        *
        * Note: a USE /D0/DEFS/OS9DEFS
        *       is between the IFP1 and
        *       ENDC statements.
                          IFP1
                          ENDC
        *
        * Equates
  0001            BUFSIZ  equ   1          Buffer size
```

```
     0001           STDOUT  equ   1          Standard ouput path
     0000           STDIN   equ   0          Standard input path
          *
          * Data section
   D 0000                   org   0
   D 0000           CHAR    rmb   1          The buffer
   D 0001           SFLAG   rmb   1          Space detect flag
   D 0002                   rmb   200        save Some stack space
   D 00C9           STACK   equ   .-1
   D 00CA           PACMEM  equ   .
          *
          *
          * program section
     0000 87CD0056          mod   PACEND, NAME, $11, $81, START, PACMEM
     000D 506163EB  NAME    FCS   "Pack"     Name of the module
     0011 01        EDITION fcb   1          Edition number
          *
     0012           START   equ   *
     0012 6F41              clr   SFLAG,U    clear space flag
          *
          * Read from STDIN a CHAR
     0014 8600      READCH  lda   #STDIN     Get from standard output
     0016 30C4              leax  CHAR,U     a character
     0018 108E0001          ldy   #BUFSIZ    one byte at a time
     001C 103F89            os9   I$read     using the OS9 Read
     001F 2529              bcs   ERRCHK     If carry is set, branch to err
          *
          * Check SFLAG
     0021 6D41      CHFLAG  tst   SFLAG,U    Is flag set?
     0023 270A              beq   CHSPAC     No, then check for a space
     0025 A6C4      HIDESP  lda   CHAR,U     Otherwise get character
     0027 8A80              ora   #$80       Set the MSB high
     0029 A7C4              sta   CHAR,U     to hide a privious space
     002B 6F41              clr   SFLAG,U    and save it
     002D 200C              bra   WRITCH     Write the new character
          *
          * Write character or set SFLAG
     002F A68D001F  CHSPAC  lda   SPACE,PCR  Get a space
     0033 A1C4              cmpa  CHAR,U     and compare it the character
     0035 2604              bne   WRITCH     If not a space, write it
     0037 6341              com   SFLAG,U    Otherwise set the flag
     0039 20D9              bra   READCH     and read another character
          *
          * Write the current character at CHAR
     003B 8601      WRITCH  lda   #STDOUT    Get the standard output
     003D 30C4              leax  CHAR,U     and the character
     003F 108E0001          ldy   #BUFSIZ    which is on byte long
     0043 103F8A            os9   I$write    and write it
     0046 2502              bcs   ERRCHK     If carry set go to ERRCHK
     0048 20CA              bra   READCH     Otherwise read another one
          *
          * Finish up the program
     004A C1D3      ERRCHK  cmpb  #211       Is this an end-of-file?
     004C 2601              bne   DONE       No, then returng with error
     004E 5F                clrb             Otherwise clear the error
     004F 103F06    DONE    os9   F$exit     and go back
          *
     0052 20        SPACE   fcc   / /        This is the space
          *
     0053 C20609            emod
     0056           PACEND  equ   *
                            end

00000 error(s)
00000 warning(s)
$0056 00086 program bytes generated
$00CA 00202 data bytes allocated
$1EE8 07912 bytes used for symbols
```

```
          ****************
          * UNPACK by Ron Voigts
          * 15-FEB-86
          *
          * This program will unpack a file
          * that was previously created by
          * PACK.
          *
          * Usage: unpack <infile >outfile
          *
                            nam   UNPACK
                            ttl   Unpacks "packed" files
          *
          * Note: a USE /D0/DEFS/OS9DEFS
          *       is between the IFP1 and
          *       ENDC statements.
                            IFP1
                            ENDC
          *
          * data section
     0001           BUFSIZ  equ   1          Buffer size
     0001           STDOUT  equ   1          Ouput path
     0000           STDIN   equ   0          Input path
   D 0000                   org   0
   D 0000           CHAR    rmb   1          The buffer
   D 0001           SFLAG   rmb   1          Space flag
   D 0002                   rmb   200        Stack space
   D 00C9           STACK   equ   .-1
   D 00CA           UMEM    equ   .
          *
          * Program section
     0000 87CD0056          mod   UEND,NAME,$11,$81,START,UMEM
     000D 556E7061  NAME    FCS   "Unpack"
     0013           START   equ   *
          *
          * Characters are read into CHAR
          * If an error occurs execution
          * goes to ERRCHK
     0013 8600      READCH  lda   #STDIN     Get standard input
     0015 30C4              leax  CHAR,U     Point to character buffer
     0017 108E0001          ldy   #BUFSIZ    Get buffer size
     001B 103F89            os9   I$read     Do a read
     001E 252A              bcs   ERRCHK     Check error if carry set
          *
          * Check bit 7 on CHAR
          * If its set a space goes before the
          * the character and bit 7 is cleared
     0020 8680      CHKBIT  lda   #%10000000 Get bit mask
     0022 A4C4              anda  CHAR,U     AND it with the buffer
     0024 2715              beq   WRITCH     If zero write it
     0026 867F              lda   #%01111111 else get mask and
     0028 A4C4              anda  CHAR,U     AND it with buffer
     002A A7C4              sta   CHAR,U     to remove reset 8th bit
          *
          * output a space if bit 7 is set
     002C 8601      SPCOUT  lda   #STDOUT    Get stadard output
     002E 308D0020          leax  SPACE,PCR  Get space character
     0032 108E0001          ldy   #1         Ready for one character
     0036 103F8A            os9   I$write    and write it'
     0039 250F              bcs   ERRCHK     Go to error if carry set
          *
          * Output CHAR
     003B 8601      WRITCH  lda   #STDOUT    Get standard output path
     003D 30C4              leax  CHAR,U     Get buffer
     003F 108E0001          ldy   #BUFSIZ    Ready for one character
     0043 103F8A            os9   I$write    and write it'
     0046 2502              bcs   ERRCHK     Go to error if carry set
     0048 20C9              bra   READCH     Go read another character
          *
          * Check for end-of-file
          * Anything else is a real error
     004A C1D3      ERRCHK  cmpb  #E$eof     End of file?
     004C 2601              bne   DONE       No, then finish up
     004E 5F                clrb             Yes, return without error
     004F 103F06    DONE    os9   F$exit     Return
          *
          * The space
     0052 20        SPACE   fcc   / /        A space'
          *
     0053 6A9613            emod
     0056           UEND    equ   *
                            end

00000 error(s)
00000 warning(s)
$0056 00086 program bytes generated
$00CA 00202 data bytes allocated
$1ECA 07882 bytes used for symbols
```

"C" User Notes

E. M. (Bud) Pass, Ph.D.
Computer Systems Consultants
1454 Latta Lane, N. W.
Conyers, GA 30207
404-483-1717/4570

Both Windrush and Microware released new versions of their C compilers....

Windrush version 27.0:0 has several significant improverments over previous versions....

The C pre-processor, has been rewritten....

The library routines have been corrected in several areas....

INTRODUCTION

This chapter provides information on recent updates to the C compilers for the 6809 and 68000. It also presents a one-screen editor written using the curses library functions.

C COMPILER UPDATES

Both Windrush and Microware released new versions of their C compilers, with Windrush C being for 6809 Flex and Microware C being for OS/9-68000. Introl notified its dealers that its 6809 OS/9 C compiler will be marketed only for Level 2, due to insufficient memory to run it on Level 1 systems.

Windrush (McCosh) C compiler version 27.0:0 has been released and has several significant improvements over previous versions.

The compiler will now run with a MEMEND less than $BFFF. This was accomplished by making the CPASS1.CMD of the compiler somewhat smaller and removing the check in CC.CMD for a MEMEND of $BFFF which was inserted in version 26, but which caused the compiler to be only marginally useful on many FLEX systems. Produced object programs will now ensure that the last address is less than the current MEMEND, to help prevent system crashes due to too-large programs.

The C pre-processor, CPREP.CMD, has been rewritten. It allows macro definitions and strings to be continued from one line to the next by preceding the intervening carriage return with a back-slash, as specified in K & R. It does its own error processing, rather than letting CPASS1.CMD flag errors in the input program, and occasionally miss them. It supports nested comments, a common extension to K & R. It detects and flags recursive macro definitions, rather than looping until it depletes its stack space.

The library routines have been corrected in several areas. Since the FLEX 1.CMD routine reverts to terminal input upon reaching the end of its input file, the user needs some manner to indicate end of file. This previously was done by entering a control-D, but version 26 of the compiler deleted this capability. Version 27 reinstates this capability. The FLEX O.CMD, P.CMD, and similar output filter commands were not processed correctly in version 26, mistakenly maintaining terminal parameters such as width and depth on these pseudo output files. This has been corrected in version 27.

A major enhancement introduced in version 27 is standard I/O redirection using the MSDOS, OS/9, UNIFLEX, and UNIX notation of "<filename" for input file and ">filename" for output file. This enhancement also corrects the problem described in an earlier chapter pertaining to the incorrect handling of quoted strings on command lines. Since this increased complexity costs a significant amount of code in the library routines, it is effective only when the user includes a dummy call to rio-init() somewhere in the C program, in a manner similar to the use of dummy calls to pflinit() or pffinit() for printf().

A known remaining problem with the McCosh family of C compilers involves the linking of variables across separately-compiled modules. Porting large C programs from UNIX, MSDOS, INTROL, or Microware OS/9-68000 C to McCosh C is almost always a challenge.

Global variables (in McCosh C) must be actually declared once and noted as "extern" in all other modules. The manner in which this is normally done is to construct a header file providing all extern variables in a common block. This header file is "#include"d in all modules. Then the variables are actually defined in one of the modules.

This procedure, for some reason, even when followed faithfully, does not always work properly. Error messages such as "entry name clash ..." or "... unresolved in ..." are sometimes generated incorrectly. Often, it is necessary to actually include the header file in each module to circumvent the linker problems. Version 27 of the McCosh C compiler has the same linkage problems as version 26 and older editions.

I do not now have the details of the changes in the new Microware OS/9-68000 C; however, I will provide them when I get them. I have learned that it is a major, not a maintenance, release, since even those with maintenance contracts must pay for it, and there are new manuals associated with it. Contact Microware for details.

CURSES ONE-SCREEN EDITOR

The power of the curses package, discussed in the last chapter, is illustrated in the C program listed below. It is a one-screen editor, using the curses functions, adapted from an AT&T-supplied set of examples for the use of curses. Although the example program is not useful in itself, the edit function in the program could be very useful in a program which manages a screen window representing a portion of a file being edited. Even if the curses package is not available on a particular machine, the edit function could be used as a model for the development of a screen-oriented editor.

```
/*  one-screen curses editor adapted from AT&T
example program */

#include <curses.h>

short int c, row = 0, col = 0;

main(argc, argv)
int argc;
char *argv[];
{
        short int i, l, n;
        FILE *fd;

        if (argc < 2)
        {
                fprintf(stderr, "Usage: editor infile
[outfile]\n");
                exit(1);
        }
        if (!(fd = fopen(argv[1], "r")))
        {
                fprintf(stderr, "Could not read
%s\n", argv[1]);
                exit(2);
        }
        initscr();
        cbreak();
        nonl();
        noecho();
        idlok(stdscr, TRUE);
        keypad(stdscr, TRUE);
        while ((i = getc(fd)) != EOF)
                addch(i);
        fclose(fd);
        move(0, 0);
        refresh();
        i = edit();
        endwin();
        if (i)
        {
                if (!(fd = fopen(argv[1 + (argc > 2)],
"w")))
                {
                        fprintf(stderr, "Could not
write %s\n", argv[1]);
                        exit(3);
                }
                for (l = 0; l < LINES - 1; ++l)
                {
                        for (n = len(l), i = 0; i <
n; ++i)
                                putc(mvinch(l, i),
fd);
                        putc('\n', fd);
                }
                fclose(fd);
        }
        exit(0);
}

len(lineno)
short int lineno;
{
        short int length;

        for (length = COLS - 1;
                ((length >= 0) && (mvinch(lineno,
length) == ' '));
                --length);
        return (length + 1);
}

edit()
{
        while (1)
        {
                move (row, col);
                refresh();
                switch (c = getch())
                {
                case KEY_LEFT: /* cursor left */
                case 'j':
                        if (col)
                                --col;
                        break;
                case KEY_RIGHT: /* cursor right */
                case 'l':
                        if (col < COLS - 1)
                                ++col;
                        break;
                case KEY_UP: /* cursor up */
                case 'i':
                        if (row)
                                --row;
                        break;
                case KEY_DOWN: /* cursor down */
                case ',':
                        if (row < LINES - 1)
                                ++row;
                        break;
                case KEY_HOME: /* cursor home */
                case 'k':
                        move(row = 0, col = 0);
                        refresh();
                        break;
                case KEY_IL: /* insert line */
                case 'o':
                        if (row >= LINES - 1)
                        {
                                beep();
                                break;
                        }
                        move(++row, col = 0);
                        insertln();
                case KEY_IC: /* insert characters */
                case 'c':
                        standout();
                        mvaddstr(LINES - 1, COLS -
20, "INSERT MODE");
                        standend();
                        move(row, col);
                        refresh();
                        while ((col < COLS - 1) &&
                                ((c = getch()) !=
0x04) && (c != KEY_EIC))
                        {
                                insch(c);
                                move(row, ++col);
                                refresh();
```

```
                        }
                        move(LINES - 1, COLS - 20);
                        clrtoeol();
                        refresh();
                        break;
                case KEY_DC: /* delete character */
                case 'x':
                        delch();
                        break;
                case KEY_DL: /* delete line */
                case 'd':
                        deleteln();
                        break;
                case KEY_CLEAR: /* redraw screen */
                case 'z':
                        clearok(curscr);
                        refresh();
                        break;
                case 'w': /* write and quit */
                        return 1;
                        break;
                case 'q': /* quit */
                        return 0;
                        break;
                default:
                        beep();
                }
        }
}
```

C BULLETIN BOARD

A bulletin board has been established by members of the Atlanta Computer Society Motorola Special Interest Group. It contains, among other areas, a public-domain C library and an amateur radio section. It currently operates at 300, 1200, and 2400 Baud. On your first call, you must register with the system operator (sysop) to be able to upload and download files. Be sure to mention that you learned of the bulletin board in C Notes. The telephone number is 404-493-4708.

C PROBLEM

The answer to the previous C problem, which was to code functions for multiplication, division, addition, and subtraction, implementing a generalized form of extended precision arithmetic, is provided as a part of the example C program below.

For the next problem, find the serious, but subtle, bug in the b-tree example program published several chapters back (six, to be more precise). [Hint: where does "gets(q)" place the string?] Provide two ways in which to fix the problem, and the advantages and disadvantages of each solution. There was also a typographical error in the original published listing, which caused a problem of its own, in function "caloc". The statement "x = sbrk(i - n - m)" should read "x = sbrk(i - n * m)".

EXAMPLE C PROGRAM

Following is this month's example C program; it implements an extended-precision reverse-polish calculator.

```
/* calculat.c - extended-precision reverse-polish
calcuator */

#include <stdio.h>
#include <ctype.h>

#define MAXPREC  128     /* data length */
#define MAXSLOT  10000   /* max value (+ 1) per slot
(ends with zeroes) */
#define MAXSLO2  16384   /* next larger power of 2 */
#define MAXDIGT  4       /* digits per slot */
```

A bulletin board has been established by members of the Atlanta Computer Society Motorola Special Interest Group.

```
#define TYPE short int  /* data type for numerics
and MAXSLOT * 2 */
#define DATA short int  /* data type for MAXSLOT * 2
*/
#define LONG long       /* data type for MAXSLOT *
MAXSLOT + 1 */

/*  structure of each extended precision number:
*/
/*      pmax  maximum number of units of precision
*/
/*      curr  current number of units of precision
*/
/*            (negative for negative number)
*/
/*      data  representation of extended number
*/
struct xprec
{
        TYPE pmax, curr;
        DATA data[MAXPREC + 1];
};

main(argc, argv)
int argc;
char **argv;
{
        char string[128];
        short int tos = -1, i;
        struct xprec reg[17], *xreg[17];

        for (i = 0; i < 17; ++i)
                pzero (xreg[i] = &reg[i]);

        while (fgets(string, 128, stdin))
        {
                switch (*string)
                {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
digits:
                        pinput(string, xreg[++tos]);
                        poutput(string, xreg[tos]);
                        printf("%02d %s\n", tos,
string);
                        break;
                case '*':
                        if (tos > 0)
                        {
                                multiply(xreg[tos -
1], xreg[tos],
                                        xreg[tos +
1]);
                                pcopy(xreg[tos + 1],
xreg[tos - 1]);
```

```
                                        poutput(string,
xreg[--tos]);
                                        printf("%02d %s\n",
tos, string);
                                }
                                break;
                        case '/':
                                if (tos > 0)
                                {
                                        divide(xreg[tos -
1], xreg[tos],
                                                xreg[tos +
1]);
                                        pcopy(xreg[tos + 1],
xreg[tos - 1]);
                                        poutput(string,
xreg[--tos]);
                                        printf("%02d %s\n",
tos, string);
                                }
                                break;
                        case '+':
                                if (*(string + 1) > 0x20)
                                        goto digits;
                                if (tos > 0)
                                {
                                        add(xreg[tos - 1],
xreg[tos],
                                                xreg[tos +
1]);
                                        pcopy(xreg[tos + 1],
xreg[tos - 1]);
                                        poutput(string,
xreg[--tos]);
                                        printf("%02d %s\n",
tos, string);
                                }
                                break;
                        case '-':
                                if (*(string + 1) > 0x20)
                                        goto digits;
                                if (tos > 0)
                                {
                                        subtract(xreg[tos -
1], xreg[tos],
                                                xreg[tos +
1]);
                                        pcopy(xreg[tos + 1],
xreg[tos - 1]);
                                        poutput(string,
xreg[--tos]);
                                        printf("%02d %s\n",
tos, string);
                                }
                                break;
                        case '=':
                                if (tos > 0)
                                {
                                        xreg[tos - 1]->curr
=
compare(xreg[tos - 1], xreg[tos], 1);
                                        xreg[--tos]-
>data[0] = 1;
                                        poutput(string,
xreg[tos]);
                                        printf("%02d %s\n",
tos, string);
                                }
                                break;
                        case '~':
                                if (tos > 0)
                                {
                                        pcopy(xreg[tos],
xreg[tos + 1]);
                                        pcopy(xreg[tos - 1],
xreg[tos]);
                                        pcopy(xreg[tos + 1],
xreg[tos - 1]);
                                        poutput(string,
xreg[tos]);
                                                        printf("%02d %s\n",
tos, string);
                                }
                                break;
                        case '.':
                                exit(0);
                        }
                }
                exit(0);
}

/*      multiply(operand1 TIMES operand2 GIVES result)
*/
/* result may not be the same as operand1 or
operand2 */
multiply(operand1, operand2, result)
struct xprec *operand1, *operand2, *result;
{
        TYPE c1, c2, d, i, j, o1, o2, sign = 1;
        LONG c, l;
        struct xprec *op1 = operand1, *op2 =
operand2;

        if ((result == op1) || (result == op2) ||
pzero(result))
                return 2;
        for (c1 = abs(o1 = op1->curr);
                ((c1 > 0) && (!op1->data[c1 - 1]));
                --c1, (o1 > 0) ? --o1 : ++o1);
        for (c2 = abs(o2 = op2->curr);
                ((c2 > 0) && (!op2->data[c2 - 1]));
                --c2, (o2 > 0) ? --o2 : ++o2);
        if ((!c1) || (!c2))
                return 0;
        if ((c1 + c2) > result->pmax)
                return 1;
        if ((o1 > 0) != (o2 > 0))
                sign = -1;
        for (i = d = c = 0; i < c1; ++i)
        {
                for (j = 0, d = i; (j < c2) || c;
++j)
                {
                        if (d >= result->pmax)
                                return 1;
                        l = (LONG)(result->data[d])
+
                                (((i >= c1) || (j >=
c2)) ? c : (c +
                                (LONG)(op1->data[i]
* op2->data[j])));
                        c = l / (LONG)(MAXSLOT);
                        result->data[d++] =
                                (TYPE)(l - (c *
(LONG)(MAXSLOT)));
                }
        }
        while ((d > 0) && (!result->data[d - 1]))
                --d;
        result->curr = d * sign;
        return 0;
}

/*      divide(operand1 OVER operand2 GIVES result)
*/
/* result may not be the same as operand1 or
operand2 */
divide(operand1, operand2, result)
struct xprec *operand1, *operand2, *result;
{
        TYPE c1, c2, d, i, j, k, m, n, o1, o2, p, q,
sign = 1;
        struct xprec *op1 = operand1, *op2 =
operand2;
        struct xprec dtemp, *pdtemp = &dtemp;
        struct xprec rtemp, *prtemp = &rtemp;
        struct xprec wtemp, *pwtemp = &wtemp;
        struct xprec xtemp, *pxtemp = &xtemp;
```

```
        struct xprec ytemp, *pytemp = &ytemp;
        struct xprec ztemp, *pztemp = &ztemp;

        if ((result == op1) || (result == op2) ||
pzero(result) ||
                pzero(pdtemp) || pzero(prtemp) ||
pzero(pwtemp) ||
                pzero(pxtemp) || pcopy(op1, pytemp)
|| pzero(pztemp))
                return 2;
        for (c1 = abs(o1 = op1->curr);
                ((c1 > 0) && (!op1->data[c1 - 1]));




            --d;
        result->curr = d * sign;
        return 0;
}

/*      divide(operand1 OVER operand2 GIVES result)
*/
/* result may not be the same as operand1 or
operand2 */
divide(operand1, operand2, result)
struct xprec *operand1, *operand2, *result;
{
        TYPE c1, c2, d, i, j, k, m, n, o1, o2, p, q,
sign = 1;
        struct xprec *op1 = operand1, *op2 =
operand2;
        struct xprec dtemp, *pdtemp = &dtemp;
        struct xprec rtemp, *prtemp = &rtemp;
        struct xprec wtemp, *pwtemp = &wtemp;
        struct xprec xtemp, *pxtemp = &xtemp;
        struct xprec ytemp, *pytemp = &ytemp;
        struct xprec ztemp, *pztemp = &ztemp;

        if ((result == op1) || (result == op2) ||
pzero(result) ||
                pzero(pdtemp) || pzero(prtemp) ||
pzero(pwtemp) ||
                pzero(pxtemp) || pcopy(op1, pytemp)
|| pzero(pztemp))
                return 2;
        for (c1 = abs(o1 = op1->curr);
                ((c1 > 0) && (!op1->data[c1 - 1]));
                --c1, ((o1 > 0) ? --o1 : ++o1));
        for (c2 = abs(o2 = op2->curr);
                ((c2 > 0) && (!op2->data[c2 - 1]));
                --c2, ((o2 > 0) ? --o2 : ++o2));
        if ((!c1) || (c2 > c1) || (compare (op1, op2,
0) < 0))
                return 0;
        if (!c2)
                return 1;
        if ((o1 > 0) != (o2 > 0))
                sign = -1;
        for (d = c1 - c2; d >= 0; --d)
        {
                for (n = 0, m = d; m < c1; )
                        pwtemp->data[n++] = pytemp-
>data[m++];
                pwtemp->curr = n;
                if ((k = compare(pwtemp, op2)) <= 0)
                {
                        if ((result->data[d] = (k +
1)))
                                for (m = d; m < c1;
)
                                        pytemp-
>data[m++] = 0;
                        continue;
                }
                for (i = MAXSLO2, j = q = 0; i; )
                {
                        pxtemp->curr = -1;
                        pxtemp->data[0] = k = j +
(i >>= 1);
                        if ((!multiply(op2, pxtemp,
pdtemp)) &&
                            (!add(pdtemp, pwtemp,
prtemp)) &&
                            (prtemp->curr >= 0))
                        {
                                q = j = k;
                                pcopy(prtemp,
pztemp);
                                if (!prtemp->curr)
                                        break;
                        }
                }
                if (q)
                        for (result->data[d] = q, n
= 0, m = d; m < c1; )
                                pytemp->data[m++] =
pztemp->data[n++];
        }
        for (d = c1 - c2 + 1; ((d > 0) && (!result-
>data[d - 1])); --d);
        result->curr = d * sign;
        return 0;
}

/*      add(operand1 PLUS operand2 GIVES result)
*/
/* result may not be the same as operand1 or
operand2 */
add(operand1, operand2, result)
struct xprec *operand1, *operand2, *result;
{
        TYPE c1, c2, d, o1, o2, s1 = 1, s2 = 1, sign
= 1;
        LONG c, l;
        struct xprec *op1 = operand1, *op2 =
operand2;

        if ((result == op1) || (result == op2) ||
pzero(result))
                return 2;
        for (c1 = abs(o1 = op1->curr);
                ((c1 > 0) && (!op1->data[c1 - 1]));
                --c1, ((o1 > 0) ? --o1 : ++o1));
        for (c2 = abs(o2 = op2->curr);
                ((c2 > 0) && (!op2->data[c2 - 1]));
                --c2, ((o2 > 0) ? --o2 : ++o2));
        if (!c1)
                return (pcopy(op2, result));
        if (!c2)
                return (pcopy(op1, result));
        if ((o1 < 0) || (o2 < 0))
        {
                if ((o1 < 0) && (o2 < 0))
                        sign = -1;
                else
                {
                        switch (compare(op1, op2,
0))
                        {
                        case -1:
                                s1 = -1;
                                if (o2 < 0)
                                        sign = -1;
                                break;
                        case 0:
                                return 0;
                        case 1:
                                s2 = -1;
                                if (o1 < 0)
                                        sign = -1;
                                break;
                        }
                }
        }
        for (c = d = 0; (d < c1) || (d < c2); )
        {
                l = c + (LONG)(((d < c1) ? (s1 * op1-
>data[d]) : 0) +
                        ((d < c2) ? (s2 * op2-
>data[d]) : 0));
```

```
                c = 1 / (LONG)(MAXSLOT);
                if ((1 -= (c * (LONG)(MAXSLOT))) < 0)
                        1 += (LONG)(MAXSLOT), --c;
                if (d >= result->pmax)
                        return 1;
                result->data[d++] = (TYPE)(1);
        }
        if (c > 0)
        {
                if (d >= result->pmax)
                        return 1;
                result->data[d++] = (TYPE)(c);
        }
        while ((d > 0) && (!result->data[d - 1]))
                --d;
        result->curr = d * sign;
        return 0;
}

/*   subtract(operand1 MINUS operand2 GIVES result)
*/
/* result may not be the same as operand1 or
operand2 */
subtract(operand1, operand2, result)
struct xprec *operand1, *operand2, *result;
{
        struct xprec stemp, *pstemp = &stemp;

        if (negate(operand2, pstemp))
                return 2;
        return (add(operand1, pstemp, result));
}

/*   compare(operand1 TO operand2 WITH signs)
*/
/* value is -1, 0, +1 if operand1 <, =, > operand2
*/
/* operand1 and operand2 unsigned if signs = 0
*/
compare(operand1, operand2, signs)
struct xprec *operand1, *operand2;
short int signs;
{
        TYPE c1, c2, d, o1, o2, s1 = 1, s2 = 1;
        struct xprec *op1 = operand1, *op2 =
operand2;

        for (c1 = abs(o1 = (signs ? op1->curr :
abs(op1->curr)));
                ((c1 > 0) && (!op1->data[c1 - 1]));
                --c1, ((o1 > 0) ? --o1 : ++o1));
        for (c2 = abs(o2 = (signs ? op2->curr :
abs(op2->curr)));
                ((c2 > 0) && (!op2->data[c2 - 1]));
                --c2, ((o2 > 0) ? --o2 : ++o2));
        if (o1 < o2)
                return (-1);
        if (o1 > o2)
                return 1;
        if (o1 < 0)
                s1 = -1;
        if (o2 < 0)
                s2 = -1;
        while (--c1 >= 0)
                if (d = ((op2->data[c1] * s2) - (op1-
>data[c1] * s1)))
                        return ((d < 0) ? 1 : -1);
        return 0;
}

/*   pcopy(operand INTO result)
*/
pcopy(operand, result)
struct xprec *operand, *result;
{
        memcpy(result, operand, sizeof(struct
xprec));
        return 0;
}

/*   negate(operand INTO result)
*/
negate(operand, result)
struct xprec *operand, *result;
{
        if (pcopy(operand, result))
                return 2;
        result->curr *= -1;
        return 0;
}

/*   pzero(operand)
*/
pzero(operand)
struct xprec *operand;
{
        int i;

        operand->pmax = MAXPREC;
        for (i = operand->curr = 0; i < MAXPREC; ++i)
                operand->data[i] = 0;
        return 0;
}

/*   pinput(string INTO operand)
*/
pinput(string, operand)
char *string;
struct xprec *operand;
{
        char *p = string;
        TYPE temp, sign = 1, ptn;

        if (*p == '-')
                sign = -1;
        for (operand->curr = temp = 0, p +=
(strlen(p) - 1), ptn = 1;
                p >= string; --p)
        {
                if (ptn >= MAXSLOT)
                {
                        if (operand->curr >=
operand->pmax)
                        {
                                operand->curr *=
sign;
                                return 1;
                        }
                        operand->data[operand-
>curr++] = temp;
                        temp = 0;
                        ptn = 1;
                }
                if (isdigit(*p))
                {
                        temp += (*p - '0') * ptn;
                        ptn *= 10;
                }
        }
        if (temp)
        {
                if (operand->curr >= operand->pmax)
                {
                        operand->curr *= sign;
                        return 1;
                }
                operand->data[operand->curr++] =
temp;
        }
        operand->curr *= sign;
        return 0;
}

/*   poutput(string FROM operand)
*/
poutput(string, operand)
char *string;
struct xprec *operand;
{
```

```
        char *p = string, form[10];
        TYPE slot = operand->curr;

        if (slot < 0)
        {
                slot *= -1;
                strcpy(p++, "-");
        }
        while ((--slot >= 0) && (!operand-
>data[slot]));
        sprintf(p, "%d", (slot < 0) ? 0 : operand-
>data[slot]);
        sprintf(form, "%c%c%d%c", '%', '0', MAXDIGT,
'd');
        for (--slot, p += strlen(p); slot >= 0; --
slot, p += MAXDICT)
                sprintf(p, form, operand-
>data[slot]);
        return 0;
}
```

- - -

SARDIS 512K RAM Expansion TECHNOLOGIES

Last year we reviewed several SBCs. One of which was the Sardis ST-2900 6809 SBC. If you remember I installed it into a Heath H19 CRT terminal, with two 5 inch drives. The system drive 80 track, the work drive a 40 track drive. It makes a very nice package. The two major problems with this system was 1.) a level one system and 2.) memory problems. However, most of that has been rectified with their latest line addition - a 512K RAM expansion, with RamDisk software.

The addition of the 512K expansion board was a real advantage. I spend most of my time programming in C and SCULPTOR+. While Sculptor+ is not at it's best on a level one system, it suffices for most of my projects. Actually I discovered that over 70% of all my Sculptor projects can be done on a level one system. As for C, well, I do most all my 6809 programming on the level one Sardis system in C. The Sardis system uses the Tandy R/S version of OS-9 and some special drivers from Sardis/D.P. Johnson. By using this version I found that the CoCo OS-9 leaves more available RAM than any other level one system I know of. So the 512K expansion has made life with level one a lot sweeter. For FLEX it is also very efficient and some of the enhanced utilities furnished make file handling very fast.

We have several different 6809 FLEX, OS-9 and UniFLEX systems with expanded RAM, up to a full meg on several, however, the Sardis 512K expansion is the least expensive of them all. And took less than a half hour to install.

The actual installation time was spent in prying apart the FDC (floppy disk controller) from the CPU board and inserting the 512K expansion board. The package is now a three board sandwich. Also we opted to install our own memory chips (256k). The price for these has fallen considerably in recent weeks. The full set of chips cost us about $45.00. And since the rest of the board had been factory built and tested everything went very smooth. The instructions were complete and easy to understand. A usual Sardis touch.

The 512K expansion uses a fully transparent refresh scheme. THe CPU accesses memory at full speed (no wait states). Also an efficient checksum routine insures the integrity of RamDisk data. This should preclude total system crashes by checking each sector as it is read or written and reporting the error immediately. Partial data corruption by line glitches, etc. will be less likely to bomb your entire work session. And of course you should run the VERIFYRO (see below) at least once a day (providing your system is never turned off) to insure nothing has clobbered part of the RamDisk.

The documentation is very complete. Full circuit diagrams, parts list and parts placement charts, PAL logic diagrams (factory and board level), technical notes and a special RamDisk memory test utility.

The RamDisk is defined to contain 16 256 byte sectors. Each fits exactly into one 4K page on the 512K board. Therefore, the board contains 128 such pages. Page 0 is used to replace the $E0000-$EFFF memory on the CPU board. Page one is reserved for storing the checksum table, leaving 126 track for a total of 2016 available sectors. FLEX automatically extends the directory as needed. Because of the high speed of a RamDisk, fragmented directories and files are of little concern.

Each sector has a two byte checksum, and since there are 2016 sectors, all the checksums fit exactly into one 4K page. The RamDisk speed is such that the checking of checksums is not noticeable and creates confidence and peace of mind in regards to data integrity. The RamDisk and the special features from Sardis make FLEX a much more efficient system.

The FLEX system (the Sardis SBC supports both FLEX and OS-9, just insert the proper disk and boot) becomes much more efficient running in RamDisk. I use the special copy command from Sardis (much faster) and load most used commands and work files into the 512K RamDisk. You cannot imagine the difference in efficiency. About every 10 minutes I backup to the work disk the current project in RamDisk (your zonked if the power flicks) and have not yet lost a single bit. I estimate that things go 5 to 6 times faster in RamDisk than on a physical disk. Just remember to make timely backups!

RAMDSK29 is the program that activates the RamDisk device drivers. It has several options. A "N" implies do not reformat the RamDisk and don't test for checksum errors. A "Y" implies yes, do reformat the Ramdisk but do not test for checksum errors. A "E" implies to reformat the disk only if checksum errors are detected. RAMDSK29 should be in the startup file, and run each time the system is brought on line. Additional options are:

TK=ddd number of tracks to allocate.
DR-d drive number to assign (default 3).

The RAMDSK29 also patches FLEX to greatly speed up the "LOADCODE" system code. If the RamDisk data is still intact reformatting of the RamDisk may not be necessary. RAMDSK29 is used by FLEX only.

VERIFYRO command (FLEX & OS-9) is used to test every sector of the RamDisk for checksum errors. Options are the same as RAMDISK29. Time to completely check all 126 tracks is approximately 8 seconds.

FCOPY (FLEX only) is much faster than the regular FLEX copy routine and makes loading the RamDisk much faster. A ratio of 2 to 7 is typical. The main limitation is that FCOPY can only copy one file per call. Options are:

D implies that FCOPY should use the current system date. If omitted the date remain the same as that of the input file.

S tells FCOPY to use a 1K buffer in order that it will fit in the UCA ($C100-$C6FF). Otherwise FCOPY will use all of user RAM as a buffer. This is necessary from programs such as calls from TSC's BASIC.

MEMTEST test all 512K bytes and uses routines from the Sardis monitor ST-Mon. By using the monitor "D F" command sequence it can run on an OS-9 system also. Test patterns and error message are displayed on the system CRT.

For the OS-9 system the disk is formatted by the regular Sardis SFORMAT. A driver RAMDSK09 and a descriptor file RO are used in normal fashion by loading. The RamDisk is defined "RO".

The net result is that the Sardis 512K expansion is a worthy addition. Once you get into the swing of having the extra ram, it becomes essential.

For additional information or to order your 512K expansion, see Sardis Technology advertising, this issue.

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

Standards

Standards are IMPORTANT, that's why I'm coming back to them again. Last time I wrote about standards I had directory structures in mind; the time before that it was terminal support. In a way I'm hitting the terminal support issue again.

Consider the Atari ST -- I think it is almost inevitable that OS-9 will be available for it soon. Consider the Commodore Amiga -- I'm not so sure about it, but I'd be surprised if you don't see OS-9 there too. Excellent! Two more large groups of OS-9 users will make a big difference. We'll see lots of new software. We'll be able to buy powerful, expandable, inexpensive computers. Our favorite operating system will be a little more widely known.

Now consider the ST again. It's got a music chip, fancy video hardware and some special keys. Customers for OS-9 on the ST will very reasonably demand support for those features. Amiga owners have machines that are even more special. The music is stereo and video hardware includes elaborate support for moving graphics. If the ST or Amiga crowd discovers OS-9 and adds hundreds of thousands of new OS-9 users to our group what will that do for us? Not much. It might even hurt.

A software company will seldom write programs for the lowest common denominator (they would never compete). They'll target a large group of likely customers and try to create something that they'll find irresistible. That means that the large group of CoCo users (and perhaps the groups of ST and Amiga users) will see lots of OS-9 software that exercises all their machines' abilities. Folks with oddball systems and ANSI terminals like me will only be able to run a little of the stuff.

We've already seen a little of this with the CoCo. Compared to the new 68000-based machines the CoCo is pretty dull, but even so there is plenty of CoCo-specific OS-9 software.

A few years ago every OS-9 user had an oddball system with his own favorite terminal. Software developers had to find clever ways to write programs that could adapt or be adapted. It wasn't easy to do, and the programs never worked quite as well as they would have if they were custom-written for specific hardware. For example, no editor on OS-9 uses more than a small fraction of my terminal's features. Back when we were an anarchistic bunch there wasn't much software available, but what there was would run on my system (or it was defective).

The Unix community is more diverse than we were, but much of their software adapts nicely to its environment. I use Unix on three very different types of computer. I use at least five types of terminal. In general, everything but the programs that use graphics work for any combination. Even some graphics programs can work through various permutations. Of course, the programs are large and make heavy use of nice standards (like termcap files).

So maybe I should get on my soap box and call frantically for standards. The idea is tempting, but foolish. It would be wasted energy. Hardware manufacturers will look for special features to distinguish their machines from others. If enough people buy the computer to make it a success by today's standards, software companies will write programs to use all its features -- including the unique ones. So much for standards.

There are some glimmers of hope. If you get a computer that supports graphics at the approximate level of the Hitachi chip much of the graphics software written for OS-9 systems may work on your system. This is particularly likely if at least two comparable machines make a big splash. Software vendors may feel moved to use the graphics standard, VDI, which Microware has adopted if it lets them sell a single program into two (or more) big markets.

The same effect may work for simple full-screen programs. Microware has defined a de facto standard for terminal description called the termset file. Perhaps it will be used for programs that don't need particularly good screen control.

The structure of OS-9 almost enforces certain standards. We are unlikely to be caught in the deadfalls that keep dropping on PC-compatible owners. OS-9 doesn't have entry points -- much less undocumented entry points. Doing physical I/O from within an OS-9 program is uncomfortable and unlikely. Device drivers and file managers can be written by anyone and used by anyone with suitable hardware. Microware has given us standards for SCF, RBF, and SBF devices. There are new standards for networks and graphics. With luck these standards will be widely supported. Even with poor luck there is a good chance that programmers will stick with OS-9's layered structure. Accessing a non-standard piece of hardware through a file manager and device driver is almost as good as strict adherence to standards.

Now I'll mount my soap box. If you decide to write software for the upcoming sexy OS-9 machines, take advantage of all the features you can, but do it within the layered structure of OS-9. First an appeal to your pride: I'm talking about good software engineering practice. Modularity and abstraction rate right up with mom, apple pie, and a work station on every desk. Second an appeal to your avarice: OS-9 is on its way up. A better machine and a bigger market are always just around the corner. If you design your software so hardware dependencies are isolated in device drivers and protocols are isolated in file managers you'll find it much easier to port your software to the next hot machine.

So hope for lots of competition in the OS-9 hardware market. Buy flexible computers if you possibly can. Stay away from non-standard features unless they are very popular or you can run without them. Best of all stick with your current system a while longer. Great things are (always) just around the corner.

You may remember that I was planning to buy a 68K system months ago. I found the OS-9/68K operating system so much better than straight OS-9 that I was going to throw all my 6809 software to the winds and move. I haven't got a new system yet. I have to admit that part of the reason is being married. The important reason is, however, that I can't make up my mind.

There are several single-board OS-9/68K systems running from the Gimix down in price and performance. They are all nice packages and the prices are nice but none of them can easily accommodate memory management. Gimix may come out with a 68020 + memory management board but it isn't here yet. I can't bring myself to spend enough to get a OS-9/68K system on a throw-away machine, and I want more out of a computer than any of the current generation of systems offer. A

well-supported expansion buss would do a lot to make me more comfortable with a computer. I could buy one now and add to it as my budget and needs dictate, but a good system with a buss is fiercely expensive. I'm still perched on the fence.

The fence is a good place to be now. If some new computer comes along and grabs a big chunk of the OS-9 market, I can jump on the band wagon and be one of the people demanding special support instead of whimpering about standards. It's easier being on the winning side.

Great Things

I thought I would have definite news about the ST or Amiga by now. I've also been sitting on the edge of my seat about an even more important OS-9 happening. I can't wait any more. This column must go in now. If the "happening" happens soon I'll send a special "extra" column to Don, perhaps he'll be able to squeeze in a few more words at (or after) the last minute. Otherwise look forward to next month. Surely it will have happened by then! Your columnist is fairly bouncing with impatience.

An Introduction To The G-64 Bus

By Cosma Pabouctsidis
CESPAC, Inc.

In recent years, the Eurocard/DIN form factor for board level microcomputer products has become the favorite choice for an increasing number of industrial OEMs. The most publicized buses using this format, however, have until now been targeted toward the high performance end of the board spectrum. These buses are usually implemented on large and expensive boards, need complex bus arbitration overhead, and are often an "overkill" for most simpler 8 and 16-bit applications. The G-64 bus, already widely pupular in Europe, was introduced in the U.S. in late 1984 by CESPAC, Inc. (Mesa, AZ). The G-64 bus fulfills the need for a compact, simple, inexpensive yet powerful, industrial grade family of microcomputer boards. With seven years of experience in the European industry, the G-64 is a proven design, backed by over 25 manufacturers offering more than 400 different G-64 modules and support software. Recent improvements to the original bus specification have been made to allow the G-64 bus to remain one of the best supports for most popular 8 and 16/32-bit microprocessors.

USING THE G-64 BUS

The G-64 bus is a second generation, processor independent, non-multiplexed 16-bit microcomputer bus aimed at low end and midrange industrial applications. The bus was first defined by GESPAC in 1979 and second sourced by France's Thomson-CSF in 1980. The bus is an open architecture and no licensing is necessary in order to build compatible products. Since its introduction in 1979, the G-64 bus has become a favorite for control applications in various industries including: semiconductor capital equipment, food processing, textile, plastic injection, and medical equipments. The bus is also widely used in several factory automation, process control and robotics applications. Also, the G-64 bus is often found in automatic test equipment, either fixed, portable or airborne, and data acquisition and remote control/monitoring systems. Because of its size and cost advantages, a G-64 bus system often serves as a front-end microcomputer to a more powerful number cruncher based on more sophisticated bus structures in a networked system architecture.

G-64 BUS ARCHITECTURE

The G-64 bus board uses the standard single height Eurocard form factor of 100mm by 160mm (4" by 6.25") which is the smallest allowed in the Eurocard specification metrix shown in figure 1. The bus also uses the 96 pin DIN 41612 pin-in-socket connector. This connector offers higher pin density and improved performance over conventional card edge connectors. The rugged Eurocard/DIN connector design of the G-64 bus allows the cards to operate with a very high level of reliability in the most hostile environments. Electrically,the G-64 bus uses five types of lines: data, address, control, interrupt, and power supply. Figure 2 shows the pin assignment. The bus is processor independent and today the 8085, Z-80, 6809, 8088, 80286, 68000, 68010, 32016 and the PDP-11/70 compatible J-11 processors are all available on the bus. The bidirectional data bus uses 16 lines to handle all data transfer, whether to memory or peripheral. The direction of the data flow is determined by a single Read/Write signal. The G-64 bus specification allows for several levels of complexity and compliance. This architecture allows for the flexibility needed to build boards that range from the simplest, and therefore the least expensive, to the most powerful, while imposing a strict guideline of upward compatibility. Figure 3 shows this layered concept. The most elementary level is the G-64's I/O map, which is in fact a subset of the main system's bus. The I/O map of the G-64 bus requires only 10 address lines, validated by a peripheral access strobe signal VPA* to decode a 1K word peripheral address space. This in turn greatly simplifies the design of a typical G-64 bus I/O card, as shown in figure 4, which permits the most efficient use of the board's real estate. This simpler architectural concept is of particular benefit to the user who designs boards specific to his application.

FULL 16-BIT CAPABILITY

In the original G-64 specification, memory is accessed through the decoding of the 16 address lines and the page line, which when validated by the VMA* signal, provides direct access to up to 256 Kilobytes on the standard bus. This addressing capability is all that is necessary for the implementation of 8-bit and simple 16-bit microprocessors. In order to properly support the most recent 16-bit microprocessors, the bus was revised in 1984 and expanded to use a 96 pin connector which provides 32 additional lines to the original

specification. These lines on the expanded G-64 bus, or G-96 bus, allow the addressing capability of the bus to be expanded to 32 Megabytes, and provide the necessary signals for the implementation of a powerful multiprocessing arbitration scheme, while guaranteeing backward compatibility with the simpler G-64 boards. At all levels of sophistication, the G-64 bus provides support for six interrupt levels; five interrupt request line (IRQ1* to IRQ4*) and one non maskable interrupt line (NMI*). The G-64 bus supports vectored and non-vectored (auto-vectored) interrupts. For vectored interrupts, the interrupting device initiates the interrupt sequence by activating the interrupt request line of the bus. When the CPU accepts the interrupt and begins servicing it, it sends out an interrupt acknowledge (IACK*) signal. When the interrupt-requesting module receives the IACK* signal, it places its interrupt vector on the data bus. The CPU uses this vector in order to quickly jump to the proper interrupt service routine. Non-vectored, or auto-vectored interrupts are needed to support the simplest and least expensive I/O modules, those modules not equipped with vector generation capability. In this case the interrupted CPU will jump to a predetermined program location. If more than one module requests the bus for interrupt at the same level, daisy-chain-in and daisy-chain-out lines assign a priority to each request based on the requesting module's position on the bus; the module closest to the CPU is given first access. Figure 5 illustrates the daisy chain's use. Other important lines of the bus are the RESET* line, and the power fail detect line PWF*. The reset line is pulled down to a low level at power-up to initialize the CPU and other parametrable modules on the bus. The power fail line is used to indicate to the CPU an imminent loss of power and to save the most valuable working parameters. The G-64 bus also supports Direct Memory Access from several sources. This DMA capability allows devices with high data throughput, such as disk and local area network controllers, to efficiently use the bus. In a typical sequence, the DMA requesting device initiates a sequence by pulling the Bus Request line low (BRQ*). The CPU grants the bus by asserting the bus grant (BGRT*) signal. Finally, the requesting device acknowledges its control of the bus by asserting the bus grant acknowledge line active (BGACK*), and keeps this line low for as long as the bus is in its control. The same daisy chain interrupt priority scheme is used for the Direct Memory Access functions. The G-64 bus caries three voltages: +5V, +12V and -12V. Most G-64 modules operate out of the +5V power supply. The +12V and -12V are primarily used to provide power to RS-232 drivers and analog converter modules. Also, the bus provides a +5V standby line to connect a battery for powering such devices as CMOS RAM or a clock/calendar.

DYNAMIC CHARACTERISTICS OF THE BUS

Timing-control and clock lines on the bus help regulate the data flow and provide timing signals to peripheral devices and memories. Depending on the microprocessor supported, the G-64 bus can operate in either synchronous or asynchronous data transfer mode. In the synchronous mode, all bus transfer occur synchronously to a master clock frequency, usually equal to a normal microproprocessor instruction cycle. The G-64 bus specifies synchronous transfer rates up to 4 Megabytes per second. The synchronous mode is supported mostly by the 8-bit microprocessors, and related memory and I/O devices. In the asynchronous mode, data transfer is totally independent of any system clock. In this mode the CPU usually initiates a transfer with a memory device and the memory in turn informs the CPU that the transfer has been successfully completed using two handshaking signals. This transfer mode allows the CPU to operate at the fastest possible speed allowed by the memory or peripheral device that is accessed. The G-64 bus will operate in asynchronous mode whenever 16-bit microprocessors are used, and the maximum speed in this mode approaches 10 Megabytes per second. Two pins on the bus will have a slightly different function depending on the transfer mode. The Valid Memory Access line (VMA*) which is used in the synchronous mode to indicate that a valid address of the memory map is available on the bus, initiates the beginning of the memory cycle in the asynchronous mode. The READY line which is used in the synchronous mode to stretch the memory cycle to accommodate slower memory devices, is used as the Data Transfer Acknowledge (DTACK*) line signaling the completion of a data transfer in the asynchronous mode. The transfer mode used mainly effects the access to memory devices; an asynchronous 16-bit CPU will require an asynchronous 16-bit memory, while a 8-bit CPU will require a synchronous 8-bit memory. Typically, peripheral devices are to be accessed in a synchronous manner, with asynchronous mode as a jumper option. The asynchronous I/O mode allows full advantage to be taken of the fast peripheral devices now available in the industry.

MULTIPROCESSING ON THE G-64 BUS

In the quest for increased system performance and speed, a recent trend is to have several microprocessors handle the task that was previously handled by one, all of them operating simultaneously on the same bus. A typical example of such an application would be the control of a robot's arm where one microprocessor would be assigned to supervise each of the arm's axis, using the common bus to communicate at very high speeds with one another. Two types of multiprocessing architecture are supported on the G-64 bus. The first relies on a Master/Slave approach, where one microprocessor card (master) supervises several other microprocessors (slaves) dedicated to an I/O function. In such an architecture, the master communicates with each slave through a dual ported memory located on the slave processor. This multiprocessing architecture has the dual advantage of being simple to implement and simple to program. Since the tasks are clearly defined for each processor on the bus, the amount of data exchanged on the same bus is concise and minimal. The most recent addition to the G-64 bus specification allows a decentralized, parallel multimaster bus configuration, whose implementation is optional. The actual implementation of the multiprocessing capability requires six lines for the priority identifier (PO*-P5*), a bus busy signal (BBUSY*) and an arbitration clock (ARBCLK). When a module wishes to take control of the bus, it places its unique 5 bit identifier code on the priority identifier lines. If another module initiates an arbitration at the same time, its code gets wire-ored on the bus. A priority resolution network resident on each CPU module, then examines the PO*-P5* lines. If the identifier found on these lines is the same as the module's, it then wins the bus. Otherwise, control of the bus goes to another module with higher priority. Figure 6 shows the logic necessary for the implementation of this powerful multiprocessing scheme. The bus priority logic is duplicated and resides on every one of the processor boards of the system. This feature eliminates the need for a central bus controller which would paralyze the entire system in case of failure. The priority level of a given processor board can be programmed dynamically by software and is therefore independent of the geographic position of the board in the backplane. The main advantage of this architecture is to offer a permanent arbitration, in such a way that a current bus master is immediately informed when there is a higher priority module requesting the bus. If the current bus master's priority is lower than the requester, the master could still keep the bus. However, by monitoring the arbitration, it would releases the bus since it has lost the arbitration. The arbitration principle allows implementation of up to 32 priority levels with "protest" feature which allows the bus requester to obtain temporarily the highest priority level. Another feature of the G-64 bus arbitration is the arbitration with "fairness", in which a module releasing the bus will not participate in the arbitration before all requesters have been serviced. In a multiprocessor system it is necessary to determine which processor is to respond to a given interrupt. In addition, it is not possible to maintain an interrupt request asserted since this would lock the bus while the interrupt is being processed. To resolve these issues, the G-64 bus supports an event passing structure. An event is a short message of two words or more which contains primarily the ID of the requesting device and the ID of the CPU involved. Event messages transfers are performed on the data bus and synchronized by the Valid Event Data (VED*) signal.

CONCLUSION

The unique features of the G-64 bus make it very attractive to an important category of system integrators. Its simple and flexible structure already accommodates most current microprocessor technologies: all-CMOS boards are being introduced on the bus, and 32-bit microprocessors will soon be available. New developments are easily adapted to the bus architecture, ensuring the G-64 a position of growing importance in the bus line-up. NOTE: * indicates an active low signal.

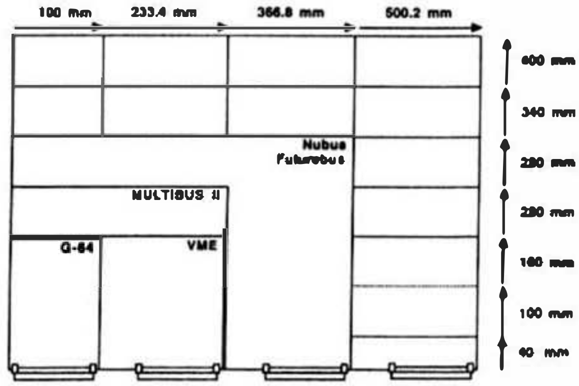


Figure 1: The G-64 bus is the only 8/16-bit bus exclusively defined for the single height Eurocard form factor.

	ROW C	ROW B	ROW A		Definition
1	GND	GND	GND	1	Power
2	A16	A8	A0	2	
3	A17	A9	A1	3	
4	A18	A10	A2	4	Address
5	A19	A11	A3	5	lines
6	A20	A12	A4	6	A0 to A23
7	A21	A13	A5	7	
8	A22	A14	A6	8	
9	A23	A15	A7	9	
10	Reserved	BRQ	BGNT	10	
11	Reserved	DS1	DS0	11	
12	Reserved	BGACK/BBUSY	HALT	12	
13	GND	Enable	SYCLK	13	Control &
14	Reserved	Reset	VPA	14	Interrupt
15	Reserved	NMI	RDY/DTACK	15	lines
16	IRQ3	IRQ1	VMA	16	
17	IRQ5	IRQ2	R/W	17	
18	VED	IACK	IRQ4	18	
19	GND	D12	D8	19	
20	P5	D13	D9	20	
21	P4	D14	D10	21	Data lines
22	P3	D15	D11	22	D0 to D15
23	P2	D4	D0	23	&
24	P1	D5	D1	24	Arbitration
25	P0	D6	D2	25	lines
26	Reserved	D7	D3	26	
27	SYSFAIL	BERR	Page	27	
28	ARBCLK	Chain In	Chain Out	28	Misc.
29	Reserved	+ 5V bat.	PWF	29	
30	Reserved	− 12V	+ 12V	30	Power
31	+ 5V	+ 5V	+ 5V	31	
32	GND	GND	GND	32	

Figure 2: The bus supports 16-bit of data and most other features expected of a modern bus.

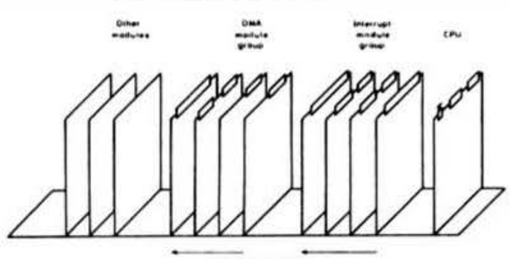


Figure 5: A daisy-chain priority scheme is used to arbitrate interrupts and DMA requests.

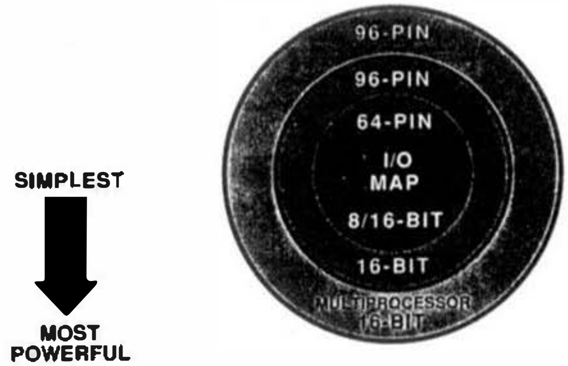


Figure 3: The G-64 bus specification allows for simple boards to be used in conjunction with the most powerful microprocessors.

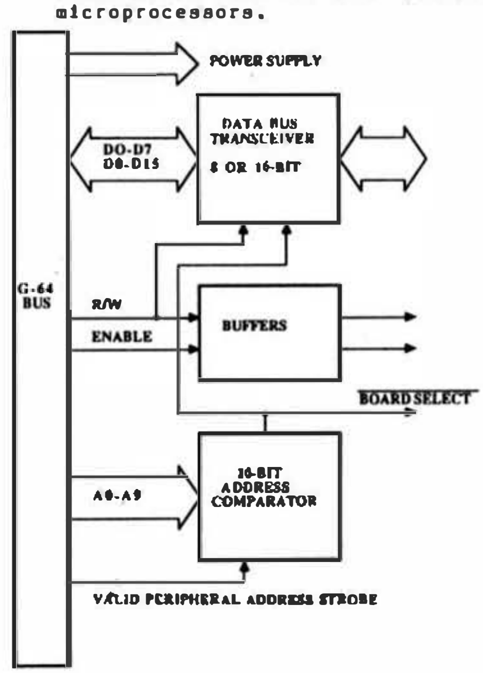


Figure 4: A typical I/O interface to the G-64 bus can be implemented in less that 5 TTL LS devices.

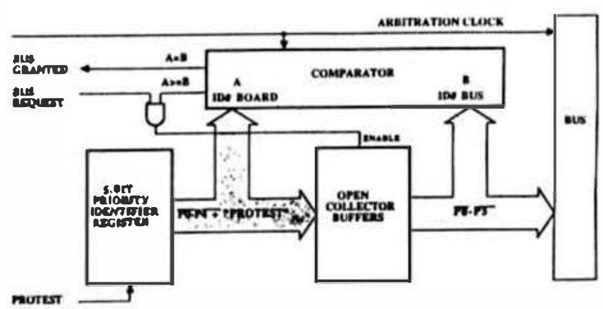


Figure 6: The G-64 bus allows for a sophisticated decentralized multi-processing arbitration scheme.

MOTOROLA

MC146805E3

Tim Ahrens
MCU Applications Manager
Motorola, Inc.
6501 William Cannon Drive West
Austin, Texas 78735-8598

One of the first entries into the CMOS family of 8-bit computers was the MC146805E2. This microprocessor provided the user with not only two on-chip 8-bit I/O ports, 112 bytes of RAM, 8-bit Timer, multiplexed Address/Data bus, but a memory map of 8k bytes. This was more than sufficient in the beginnings of the "minimum chip count" systems, when CMOS EPROMS and masked ROMs were of relatively small memory size.

This MPU became the cornerstone of Motorola's CMOS architecture, with an efficient Instruction set, true Bit Manipulation, and 3 to 6 volt operation. When integration of all members of a system became viable in the CMOS process, the E2's structure was carried into the MC146805G2, MC146805F2, and EPROM versions of the same devices.

In the early beginnings of MCU code generation, it was thought that a memory map of 8k would be sufficient for most controller type of applications. It was also thought that more than 8k would not be necessary for most applications. Since the 'E2 was expandable, many applications started using only segments of the 8k address space. Later, EPROMS became more inexpensive, marketers promised more features, and software writers were pressed into code compaction scenerios on almost every project.

For many applications, a large address space full of ROM or EPROM may mean either inefficient code generation, or a product with features which may overwhelm the processing power of the host MCU. On the other hand, there are applications which have neither of the above afflictions.

Take for instance an application which uses over 16k of ROM, but still does not tax the capabilities of the 6805. A recently designed HF radio transceiver has features which include: Band Scan, Memory Scan, Real Time Clock, Teletype and CW decode, direct frequency entry, Optical Encoder tuning, and remote control via RS-232. Most of these functions are mutually exclusive, and have little interaction upon one another.

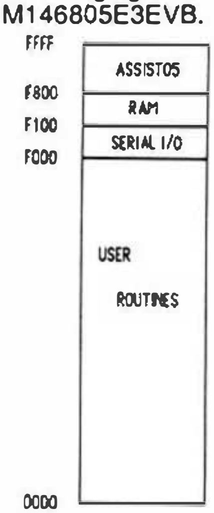
Initially, the project was started with the MC146805E2 as the host microprocessor. The further the project went into design, it became clear that the 8k map of the E2 would not be sufficient for all of the required functions. Since a considerable amount of effort had already been spent on software generation, it was not feasible to change host MPUs. Other CMOS MPUs which had an expansion bus were not available either. A simple method of expansion used since the early times of computers was incorporated into the radio. Map switching may be bothersome, but by specifying the entry and exit points of each map as well as parameters removes some of the frustration. Certainly, an MPU with a larger memory map would be desireable.

Enter the MC146805E3

In a second source agreement of transfer of technologies, Motorola received the rights and necessary data bases to manufacture the MC146805E3, a 64k map version of the original E2. The real differences in the two parts are the lack of three I/O bits on Port A. These lines have become A13, A14, and A15.

With most MCU parts, Motorola develops 'monitor' types of programs which allow users to evaluate the MCU without having an expensive development system. The E3 is different, having no on chip ROM since it is an MPU. In an earlier effort, Motorola published the external monitor program 'Assist05', which was used in an expanded MC146805F2/G2 emulator which used the E2 as a host processor. This monitor has been modified for use with the E3, and has some additional caoabilites

The following figure shows the actual memory map of the M146805E3EVB.



Emulation Techniques

There are several different methods which have been used to emulate and evaluate Motorola style Microprocessors. I will describe two methods in this article.

The first method actually provides two memory maps of 64k each. One map is the users, and has no restrictions on its use. The second map is where the monitor ROM, temporary RAM, and serial I/O ports are located. Since a full 64k is available to the monitor, as much ROM may be used as is required, thus providing many functions such as assemblers and disassemblers, multi-level breakpointing and tracing activities. The monitor program provides for two RS-232 ports - one for a terminal, and one for a host computer. Through either of these ports a program may be downloaded into the user's memory map. This users map also has a full 64k map, which allows a program to be placed anywhere.

The heart of this style of system is the map switching logic. Relying on a predictive status line of the MPU which tells the monitor when an op-code fetch is taking place, this logic actually forces a Software Interrupt onto the Data bus. When the interrupt service routine is entered, appropriate action is taken, depending on where the SWI originates (User's code, or by map switch). This type of emulation is done on the M68705, M1468705, M68701, and M68HC11 EVMs. They are called Evaluation Modules rather than emulators because not all chip specifications are truly emulated, and they are of relatively low cost.

The second method, which really cannot be considered true emulation is accomplished by actually taking a 'small' portion of the memory map, using it for monitor routines, temporary RAM, and serial I/O ports. In this type of 'emulation', the user must relocate his code to a lower portion of the memory map. Since the 68XX style of architecture has its interrupt and Reset vectors located at the top of memory, all monitor programs must reside there. Because of this, true emulation cannot be accomplished. Any interrupt service routine will generally have additional time added to it, because of the intermediate jumping which must be directed through temporary RAM.

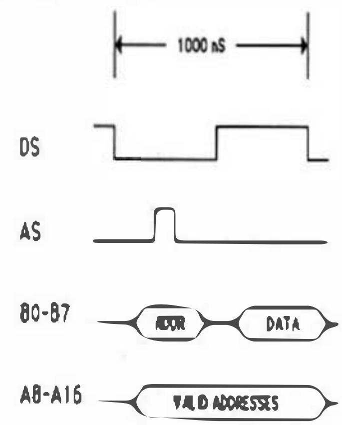
There are many things which must be considered in the selection of these two different types of systems. First, some MPU/MCUs do not have the necessary lines to determine when to switch maps. This may be accomplished by a sophisticated type of bus monitor, leading to many additional parts such as fast bipolar PROMs and extra 'glue' parts. Certainly, the number of parts is directly related to the cost of the product! This cost is a direct factor on which product is to be used. This article deals with the second type of chip evaluation, and requires very few extra devices. The schematics shown in Figure 2 are all that are required to make the E3 Evaluation Board.

System Operation

The heart of the EVB is the MC146805E3 MPU. The multiplexed Address/Data bus is de-multiplexed by a HC373. This 373 is all that is required to provide a full 16 bit Address and 8 bit Data bus for the user.

Note that only the lower eight address lines are multiplexed with the data bus. This allows the upper eight lines to be used with 'slower' decoding devices such as PROMs and discrete logic without the worry of propagation delay problems.

Like all 68XX types of MPUs, a synchronous bus is used to transfer data between various I/O and Memory devices. Basically, one signal on the E2/E3 provides system synchronization. This signal is called Data Strobe (DS), and is analogous to the earlier E or 02 signals of the MPU's parent family. During DS low time, addresses are being setup, and other signals are settling. This time period, in conjunction with the Address Strobe (AS) signal, is when the actual bus de-multiplexing takes place.



Although the schematic shows a 5 MHz crystal used in the MPU's oscillator circuitry, the system may be run slower. Alll Motorola's CMOS and HCMOS computers are fully static, and can be run from DC to full bus frequency. On the E2 and E3, the crystal frequency is divided by 5 to provide the necessary bus frequency. Certainly, if the user requires it, an external oscillator may be used to provide system timing.

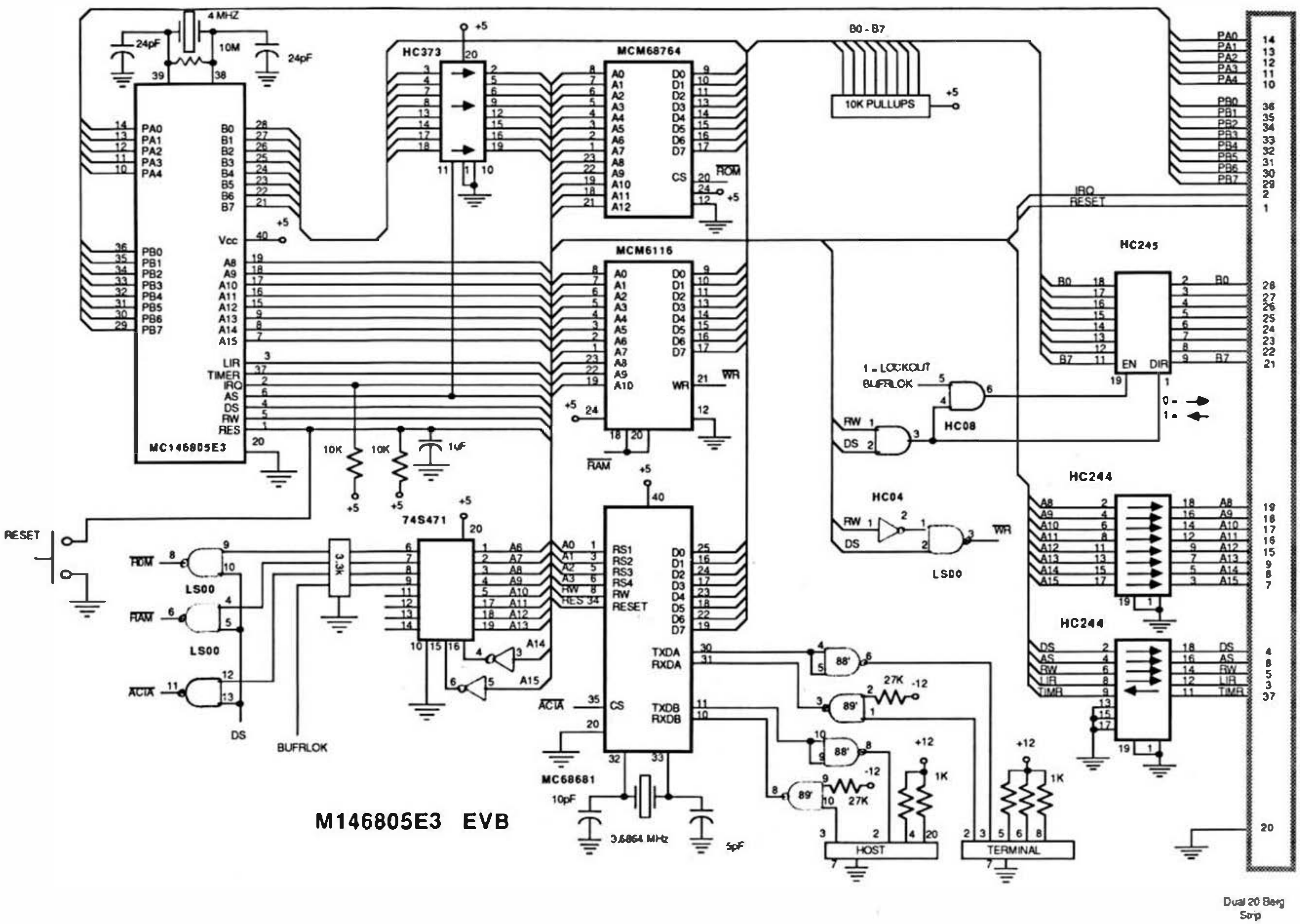
Since the E2/E3 uses a memory mapped technique to address all I/O ports and memory, all external devices must have their own chip select lines. In the E3EVB, these selects are provided by a 256x8 bipolar PROM. A PROM was chosen because it represented the fewest number of chips necessary functions. If more features are to be added to ASSIST05E3, or user routines are to be placed in EPROM, the select table of the PROM may be easily changed to suit the user's needs.

Memory mapped I/O is quite simple to use, as most instructions work with I/O, as well as normal memory instructions. Take for instance the following example which will take one byte of data from RAM, and transfer it to the data register of an I/O port.

```
LDA MEMORY1     Get Data from RAM
STA IOPORT      Put data into the Port
```

The 6805 family, like that of all other Motorola processors use this technique to easily access any type of I/O port or Memory, without having to learn specific I/O types of instructions.

The rest of the schematic is detailed with ROM, RAM, and a MC68681 DUART which provides the dual serial ports. Bus buffers are provided to isolate the users circuitry from the rest of the EVB.


M146805E3 EVB
MC146805E3
HC373
MCM68764
MCM6116
74S471
MC68681
HC245
HC244
HC244
HC08
HC04
LS00
10K PULLUPS
B0 - B7
4 MHZ
10M
24pF
3.6864 MHz
10pF
5pF
RESET
1 = LOCKOUT
BUFRLOK
HOST
TERMINAL
Dual 20 Berg Strip

ASSIST05E3

The ASSIST05E3 monitor program is intended for use with the MC146805E3 Microprocessor. The monitor uses an RS-232 link which allows a user to quickly perform both hardware/software development as well as system evaluation. A second RS-232 link is provided for use with a host computer for download capabilities. ASSIST05E3 includes commands which allow for memory and register examination/change, breakpoint set/display, single or multiple instruction trace, a transparent mode for host communications, and a command for downloading from a host computer. The following short paragraphs will explain each command. Also, ASSIST05E3 uses only external RAM for it's temporaries, thus leaving most of the on-chip RAM free for the user.

R - Register Display

The processors registers are displayed.

A - Display/Change the Accumulator

This command begins by printing the current contents of the Accumulator in Hexidecimal, and allows the user to changes it's contents

X - Display/Change the Index Register

This command does the same type of function as the "A" comand, except that it functions on the Index register.

C - Display/Change the Condition Code Register

This command does the same type of function as the "A" command, except that it functions on the condition code register.

P - Display/Change the Program Counter

This command does the same type of function as the "A" command, except that it functions on the program counter. This command is used mainly for tracing.

Breakpoints

Up to three breakpoints may be used to allow debugging of user programs. Program execution may be halted at specified addresses so that the current registers and memory may be examined. Breakpoints set for non-RAM locations will generate an error message. Whenever program execution reaches a breakpoint address, program execution ceases, the current registers are displayed, and the prompt character is returned. Following this, appropriate breakpoint commands may be entered.

B - Display Breakpoints

This command allows all breakpoint addresses to be displayed.

B N XXXX - Set breakpoint N

This command enables breakpoint N, where N is a number 0-2 at address XXXX, where XXXX is the address of the last instruction to be executed.

B N O - Clear Breakpoint N

This command disables breakpoint N, where N is a number 0-2.

Instruction Trace

This command is used to execute one or more instructions, and is generally used after a breakpoint is reached. Tracing may also be used to step through ROM-based programs: however, unlike breakpoints, tracing is not done in real-time. To use the trace command on ROM based programs, the user must put a jump-to-the-ROM entry address in RAM. The user then sets a breakpoint at the jump instruction address. Once the breakpoint address is encountered, the jump is executed and control is returned to the monitor program. The current program counter (PC) then points to the ROM entry address, and tracing may then be used.

T XXXX - Trace Instructions

With this command, XXXX instructions are executed, beginning with the current PC. If XXXX is not specified, only one instruction will be traced, with registers being displayed.

M XXXX - Memory Examine/Change

With this command, all memory locations of the MC146805E3 may be examined and changed.

G XXXX - Execute user Programs

This command allows the user to execute programs which have been downloaded or manually placed in RAM. If XXXX is not specified, the program will begin execution from the current program counter.

H - Host Communications

This command allows communication with a host computer connected to the HOST RS-232 connector. A Control "A" ch
aracter will return to ASSIST05E3.

D - Download from Host port

This command allows the user to type any character string which will be echoed to the computer connected to the Host port. After a carriage return is entered, ASSIST05E3 waits for the Host to send down S1-S9 records which will be loaded into RAM.

Although the M146805E3EVB with ASSIST05E3 has been constructed with a minimum number of external components, it has capabilities which turn it into a viable development tool. A 40-pin ribbon cable from the bus side of the schematics in Figure 2 to the user's target system is all that is required to allow the EVB to start doing useful

development work. For downloading into the memory map, the user must replace any ROM/EPROM that may be on the target system with RAM, so that S1-S9 records may be downloaded into it.

For systems that require not only CMOS technology, but a relatively large address space, the MC146805E3 may fill the bill for many users.

Versatile Chip Set Puts Overlay In Your Graphics Design

Patrick J. Svatek

and

Geoffrey Perkins

MOTOROLA INC.

6501 William Cannon Dr. W.
Austin. Tx. 78735-8598

Many video graphics applications require simultaneous text and graphics to be presented on a display screen that has an external source of video information present. Overlay applications typically occur when a camera, video disk or video cassette recorder output requires additonal information to make the picture more useful.

Where are video overlay applications likely to occur? Applications like medical imaging, video micrometry, industrial inspection systems, complex animation, security systems, environmental systems, such as undersea camera work, video printing systems and video disk based recording or educational systems can all use overlay. Each of these systems will place different demands on the text and graphics generating system.

The fundamental problems encountered by designers of microprocessor based overlay graphics systems can be catagorized into five major areas: (1) The Microprocessor interface, (2) the memory interface, (3) the video interface, (4) the overlay interface, and (5) the software overhead in generating text and graphics. Additonally, memory bandwith, CRT limitations, and , of course, performance/price ratio will influence the design.

The solution to this problem is to create a VLSI graphics system in silicon that fixes the hardware architecture, thus minimizing hardware design time, yet allows enough flexibility for the system designer to choose the amount of memory, horizontal and vertical resolutions, the color selection, the choice of microprocessor and the video interface. The next step is to make the system highly progammable so that the design can easily be adapted to a multitude of applications.

The Raster Memory System (RMS) is a two chip set that generates video displays for a full range of performance. The RMS consists of the MC68486 Raster Memory Interface (RMI) and the MC68487 Raster Memory Controller (RMC). The versatility of the RMS lies in its capability to support many modes of operation with minimal hardware impact and the implementation of software tools in silicon, resulting in a significant reduction in microprocessor overhead in generating text and graphics. A designer starts by defining a hardware system and then proceeds to choose the appropriate video mode. Then the system performance is determined by the choice of microprocessor, the amount of dynamic RAM and the level of sophistication of the software.

Several features have been included into the RMS to ease the designer's task. As shown in Figure 1, the RMI provides the master oscillator for the system. It supplies the RMC with three timing clocks, the microprocessor clock, generates multiplexing signals (ADEN,ADSEL,and DBEN) for display and MPU accesses to memory, provides DRAM address generation including transparent DRAM refresh, and decodes addresses in the memory map other than its own. The additional address decoding brought out on the 3-bit S bus allows chip selection functions to be implimented with a 3 to 8 line decoder, thus providing a system and seven more chip selects for peripherals. The X bus shown in the block diagram is used for address multiplexing as well as a data path for the RMC to communicate with the RMI.

THE MICROPROCESSOR INTERFACE

Since the RMS operates with any of three popular microprocessors -the MC6809E, the MC68008, and the MC68000- the RMI will provide Data Transfer Acknowledge (DTACK) handshaking and a 7.95 Mhz MPU clock if a 68000 family MPU is used or Enable (E) and Quadrature (Q) in a 6809E mode. It is the designers choice to have the Raster Memory System generate DTACK for some or all of the system peripherals. The only requirement is

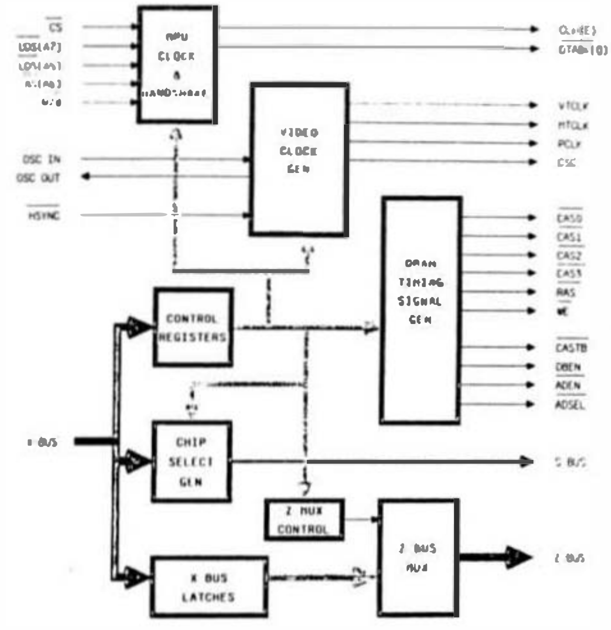


FIGURE 1
RASTER MEMORY INTERFACE BLOCK DIAGRAM

that the peripheral be fast enough to respond to DTACK within the time frame allotted by the system. Any peripheral that does not use the handshaking signals must either inform the 68000 that it is a 6800 family member and will respond synchronously or must furnish its own DTACK.

The other microprocessor interface signals , Address Strobe (AS), Upper Data Strobe (UDS), Lower Data Strobe (LDS), Read/Write (R/W) and Data Bus ENable (DBEN) complete the microprocessor interface. The AS line is a handshaking signal that indicates to the system that the address on the processor bus is valid and that the processor is ready to initiate a machine cycle. UDS and LDS have two functions: The processor uses them to indicate whether the current cycle involves the high data byte, the low data byte, or both, of a 16 bit transfer. The UDS and LDS signals are also used to inform the RMS that the current processor cycle has ended. R/W connect to both chips and informs them of data direction flow on the data bus. DBEN is used by the RMC for gating the MPU data bus onto Ports A and B of the RMC or DRAM.

THE MEMORY INTERFACE

The system works with several types of dynamic RAMs. A designer can select memory size which can be as little as 16 kbytes in a low end graphics system or as much as 1 Mbyte in the largest system configuration. The type of DRAMs that the system can use are 16k by 1, 16k by 4, 64k by 1 and 256k by 1 and should have a 150 ns access time for proper system operation. The system designer has a choice of one, two, or four banks of DRAM. With a single bank, there will be a loss in performance since the RMS uses the page modes access features of DRAM to improve the data throughput rate. In a 68000 based system, the memory is configured in either two or four banks. Since the memory is not solely used for the display process, cost effective systems can utilize another portion of memory for program storage, stack or scratch pad RAM.

The system operates synchronously to maintain a steady stream of video information to the screen. This basic system timing, called a memory cycle (9 MTCLK cycles), imposes limitations on the microprocessor throughput. In a 6809E system, the MPU is run synchronously with this timing, but in a 68000 type system, the RMS adds two wait states (250 nsec each) to every processor memory access. Additionally, the processor will at times request data at an instant that the system is unable to provide it. While these are limitations, it is unlikely that the design will require a faster version microprocessor because the video memory manipulation capabilities of the RMS substantially reduce the microprocessor's overhead tasks.

During each memory cycle, approximately 1 usec, access to DRAM is multiplexed so that the microprocessor and the display process each get one access. In 8 bit bus systems, microprocessors are allowed to read or write one byte, while a MC68000 can access either a byte or a 16 bit word. The number of bytes fetched by the display process will depend on the number of memory banks used and the type of memory cycle the current access requires. Different types of memory cycles are used by the RMS to handle transparent DRAM refresh, fill list buffers, fill object buffers and arbitrate the type of microprocessor memory access. Data called from RAM is latched into to the RMC via Port A, shown in Figure 2, (and Port B for 16-bit systems only) with a composite CAS strobe (CASTB) generated by the RMI. This raw data will be internally pipelined and processed until it emerges as analog RGB video signals.

THE VIDEO INTERFACE

The Raster Memory Controller, shown in Fig. 2, contains all the circuitry for generating the video timing signals. It also generates all display addresses, passing them to the RMI on the X-bus; receives data from memory; processes that information into pel by pel video information; maintains the DRAM refresh addresses, contains all the registers for programming the RMS, and has an on board display Arithmetic Logical Unit (ALU) that performs real time calculations for screen data manipulation.

One of the most difficult problems encountered in a microprocessor/video design is working out all those stringent timing details. In converting digital data to video, the RMI provides,a video timing clock (VTCLK), a memory timing clock (MTCLK), and a Pixel clock (PEL) to the RMC as shown in the lower left portion of the block diagram. RMC maintains synchronization with the RMI by providing a horizontal sync (HSYNC) reference pulse back to the RMI. Thus all the video timing is defined for the system designer. The designer sets the mode of operation, NTSC or PAL and 6809E or 68000 family, by simply connecting the appropriate X bus line to the system reset circuitry. The RMS maintains proper signal timing for all aspects of the system by clock stretching MTCLK, if necessary, at the end of a memory cycle and re-sychronizing both MTCLK and PCLK to VTCLK at the trailing edge of horizontal sync. VTCLK is a free-running clock and it is never

re-sychronized, while PCLK and MTCLK have their frequency defined by software selecting the horizontal resolution. PCLK varies from 6 to 14 Mhz, while MTCLK is always the master oscillator divided by four plus any stretching for a ninth cycle of the basic memory cycle.

The RMS has programmable horizontal and vertical resolutions allowing the same program to run on a variety of displays. Assume a program created using a high resolution work station monitor must be displayed on a lower resolution system or a home television. If the virtual screen is defined to be the same on both systems, the same data base can be used. The lower resolution system would simply display less of the data at one time and scrolling could be employed to view all of the original screen. There are four horizontal resolutions, 256,320,512, and 640, and up to 10 vertical resolutions, ranging from 192 to 500 lines, offered within the system. Not-interlaced, interlaced sync and interlaced sync and data modes are also software selectable.

applications, it creates many subtly different shades of the same color. In others, it can generate a large selection of divergent colors. With 12 bits defining color, the RMS has a color pallette of 4096 colors. The color selection is limited by the mode of operation: 16 colors in bit plane and 32 colors in list modes. True objects use the color mapping RAM somewhat differently in that they include a transparency option and up to 24 different colors for each object.

Another important tool of the system is the virtual screen, which is used when a picture will not fit onto the displayed screen without a loss of detail. In conventional video graphics systems, the programmmer stores a complete picture in one section of memory, then transfers a subset of that memory to the screen display memory. The technique allows a operator to pan through the picture, concentrating on individual sections. This continual data movement is microprocessor dead time if it is also burdened with complex calculations while it is trying to do screen management

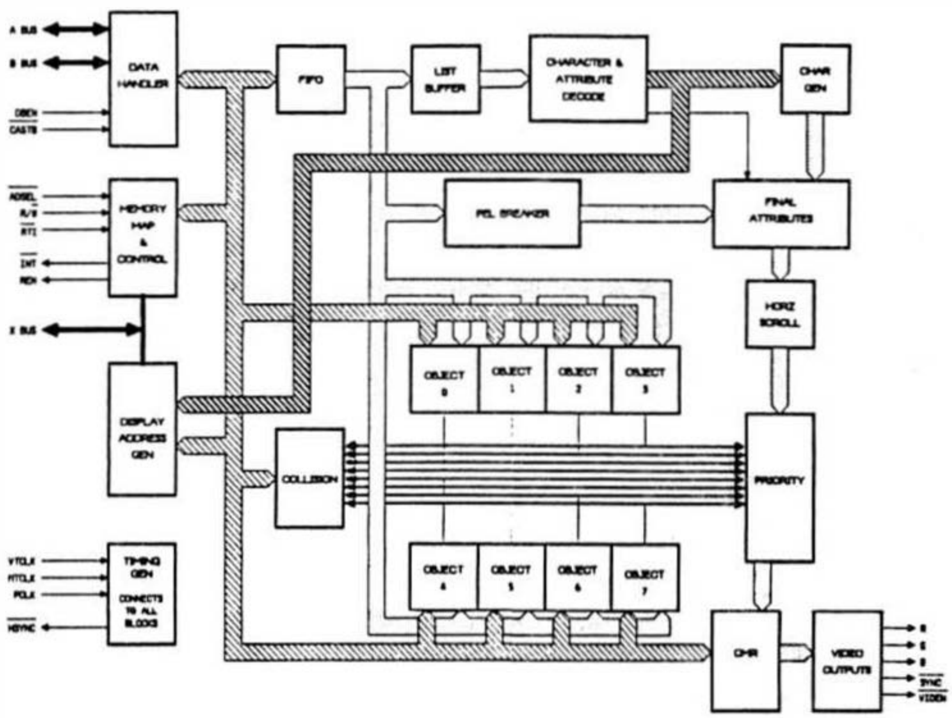


FIGURE 2 RASTER MEMORY CONTROLLER BLOCK DIAGRAM

SOFTWARE TOOLS PUT IN SILICON

All data that is processed within the system results in a 5 bit value that addresses a programmable color look up table or color mapping RAM (CMR) within the final stages of the RMC. The CMRs consist of 32 word locations. The contents of each of these 16-bit locations define a unique color. Twelve of the bits -4 bits each for red, green, and blue can be programmed directly by the microprocessor. An additional bit within each CMR brought out to the VIDEN pin on the RMC. It is used for pixel by pixel switching capability in overlay applications. The color mapping RAM is a powerful tool for controlling a video screen. In some

functions; not to mention operator frustration each time large amounts of data are moved.

The RMS eliminates the microprocessor overhead for large data block movements by supporting a virtual screen in hardware. In standard systems, hardware defined video memory is scanned in a linear fashion, thus the microprocessor has the burden of adjusting that memory with the correct data. The RMS ALU makes real time calculations for each memory cycle fetch of data, thus the displayed memory does need not be manipulated to do scrolling or paging of data.

To use the virtual screen, a programmer starts by defining the height and width of the virtual screen using a few control registers. Then the display screen size is selected by programming the horizontal and vertical resolution. Finally, the address of where the displayed screen starts in the virtual screen is programmed. Once these registers are set, the programmer now has some scrolling options. For one, he can use the smooth scrolling feature of RMS to pan the virtual screen without having to move any bytes of display data. The RMS offers pixel by pixel scrolling both horizontally and vertically for all display modes. A second option might be to use wraparound scrolling, where the display memory is treated as a toroid. That is, if scrolling proceeds long enough in one direction, it will eventually end up at the scroll start location. Yet, another method of scrolling is to treat the screen as a rectangle. When scrolling reaches a border on the rectangle, the screen is filled with a user defined constant, indicating an off screen event.

SELECT THE CORRECT DISPLAY MODE

In converting memory data to pixels, the RMS supports two basic modes: bit-plane mode, where the user can display individual pels, or six text, character, graphics or games modes called list modes. Additionally, independent of the mode, their are eight identical sets of registers to allow simultaneous independent operation of small X,Y positionable windows called true objects. True objects are hardware intensive objects that move around the screen. They have the capability for collision reporting with fixed or other true objects and priority which allows simulating three dimensional effects all defined in hardware.

In the bit plane mode, data in the virtual screen (and display screen) is converted directly to pixel data. (The screen data is always a CMR address). The programmer has the option of selecting 1,2 or 4 bits per pel which in turn determine whether 2,4 or 16 colors can be used, respectively. Each increase in bits per pel (BPP) results in a doubling of memory requirements; a 320 by 210 resolution screen requires 8.4 kbytes at 1 BPP; 16.8 kbytes at 2 BPP; and 33.6 kbytes at 4 BPP. In the bit plane mode, pixel by pixel independence is obtained at a cost of memory and when more memory must be manipulated the resulting drawing rates decrease.

The list modes of the system were designed to use memory efficiently by giving up some but not all of the pixel independence features. List modes can be thought of as indirect addressing modes, where the data fetched from display memory is used as a pointer to another memory location where the pixel data is stored. The pixel data then contains the CMR address for the pixel color. The same 320 by 210 screen described above can be represented by 3.2 kbytes in list mode 4 with 16 colors. This dramatic reduction of memory results when pixel patterns are well defined, such as text and characters. When pixel patterns are more character oriented, the system allows programming of attributes for further character enchancement.

The list modes were optimized to be oriented toward specific applications. Games and animation applications , text and word processing, mixed text and graphics modes, and graphics modes are all supported. Attributes or memory size versus performance trade offs are made by simply choosing the right list mode. For example, list mode 0 offers very little in the way of attributes, but its advantage is the small memory requirements because each byte describes one character. In list mode 4, all three bytes describe one character. This mode mixes several types of characters and offers all the RMS attributes that apply to mixed text and graphics applications or games. List mode 5 is designed for high resolution text applications, primarily word processing, but it also has sufficient graphics capability for other applications.

The programmer must be more careful when using list modes because the display memory may not be contiguous. In the list modes, data from display memory for character description will be one, two, or three bytes. When more than one byte is used to define a character, the additonal bytes will define attributes, multiple characters, multiple characters with limited attributes or one character with extended attribute capability. All the standard terminal attributes like underline, multiple flash rates, invert, double high, and double wide are supported in silicon. Graphics attributes supported include CMR offset, color/resolution selection, foreground/background color selection, and mosaics 4/6 with separation. The true object attributes include priority, color collision reporting, transparency, independent collision enables, and shading. The attributes used for a display screen are fixed by the list mode used for character attributes, while the true object attributes in general are more independent of mode selection.

Three different types of characters are supported by the RMS: alphanumeric, mosaics and redefinable characters. Alphanumerics are generated by an on-board ROM that appears invisible to the microprocessor. Internal routing on the RMC automatically selects the ROM when required. The additional bytes of decoded display data is routed to the attributes section of the RMC. Mosaics are used to create crude graphics with very little memory requirements. Mosaics are allowed to be either 4 or 6 blocks per character size. Actual size of each of the mosiac blocks may vary from 2 to 5 scan lines in size depending on the size of the character block size selected. Character height is programmable for all list modes with 8,10,12 or 16 lines per character available.

Redefinable characters are essentially RAM based characters that allow a programmer to define a custom set of pixel patterns. These characters have their pixel patterns stored in image tables. The programmer determines the size of each of the patterns, calculates the memory requirements based on the resolution and bits per pel used and stores the patterns in RAM. Memory requirements for each character can range from 8 to 64 bytes. Once the first byte location of the first character is programmed into the RMC, the ALU will make all calculations for the pattern fetches, while the programmer only has to refer to them by number. Thus some of the pixel independence of bit-plane is retained and a much easier method of using them in software is available. To support custom character sets, like Kanji, or specialized graphics characters, the RMS allows creation of up to 32,000 redefinable characters in list mode 2.

True objects are hardware intensive objects that move around the screen. In hardware, they have the capability for collision reporting with fixed or other true objects and priority which allows simulating three dimensional effects. Fixed objects are redefinable characters with additional attributes for interaction with true objects. Objects are defined in image tables like redefinable characters. Moreover, the objects can have a base of 3 colors plus transparency for every pixel and there is color offset allowed for each pixel, resulting in 24 usable object colors. Objects can be defined to be from 14 to 49 pels wide by 31 lines and can be expanded by hardware zoom by a factor of 1,2,4 or 8 in both X and Y directions independently.

Overlay with the RMSthe MC1378

To make the RMS operate synchronously with an external video source consider the addition of the MC1378, video overlay sychronizer (VOS), shown in Figure 3. This device contains a complete encoder, i.e. subcarrier frequency to the VOS. It also controls the Remote/local switch pin for selecting the sychronizing source and the Video Enable pin on the VOS for pixel by pixel switching between the video sources. The vertical/composite pin functions as either an input or output pin depending on the selected mode.

In a local mode, the RMS provides all the sychronizing signals, except the 36 Mhz which the VOS always provides to the RMS. In this case, the RMS provides composite sync via the vert/comp pin. Phase detector 1, PD1, in the VOS (see figure 3) locks the internally counted-down 4 Mhz Voltage Controlled Oscillator (VCO) to the RMS horizontal sync (HSYNC pin). PD2 and PD3 are not used in the local mode. PD4 is active providing an arbitrary phase shift between the color oscillator and the output burst phase. PD5 locks the 36 Mhz RMS clock to the 14 Mhz color oscillator. Therefore the 14 Mhz oscillator is the system standard in the local mode.

In the remote mode, the incoming video

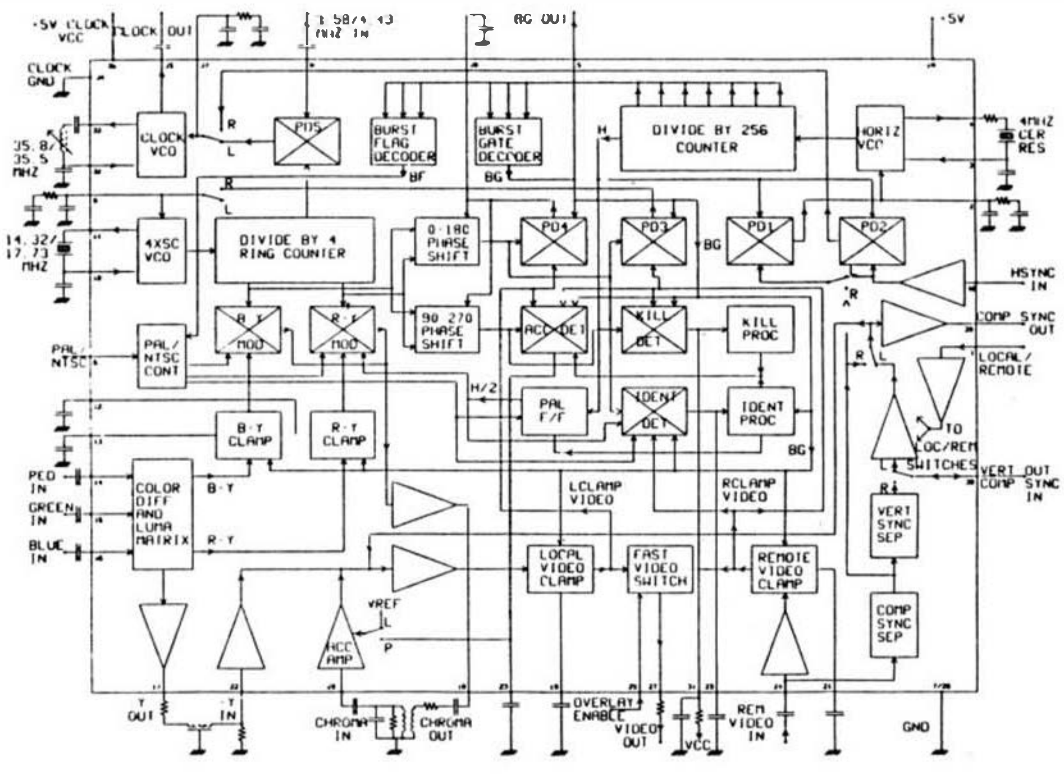


Figure 3 MC1378 COMPOSITE VIDEO OVERLAY SYSTEM

quadrature color modulators, RGB matrix, and blanking level clamps, plus a complete complement of sychronizers to lock the RMS video outputs to any remote video source. It can be used as a local system timing and encoding source for the RMS or it can be used in the overlay (remote) mode.

The block diagram in Figure 4 shows the interface between the RMS and the MC1378. Basic system timing for the RMS is derived from the 36 Mhz oscillator on the VOS. The exact frequency is either 8 or 10 times the color burst frequency depending on whether PAL or NTSC timing, respectively, is used. The RMS provides RGB, Horizontal sync, color signal supplies all the sychronizing information. A phase lock loop (PLL) is locked to the separated composite sync from the remote signal (PD1, horizontal VCO and 256 divider/counter). This produces a noise protected horizontal reference signal. Vertical lock is obtained by continuously resetting the sync generator in the RMC with a separated vertical pulse also obtained from the external video signal. The 14 Mhz VCO is phase locked to the external color burst frequency using a divide by 4 counter and PD3. Phase detector 4 controls an internal phase shifter (0-180 degrees) to assure that the outgoing color burst phase is the same phase as incoming burst phase. The internal

New Software Additions

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

New Software Additions

KANSAS CITY BASIC

KANSAS CITY BASIC - Basic for Color Computer OS-9 with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCo OS-9 $39.95

LSORT

LSORT - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.

OS-9 $85.00

X-TALK

A C-Modem/Hardware Hookup

X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTP 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020.

The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020.

The X-TALK software is furnished on two disks. One eight inch disk contains the S.E. MEDIA modem program C-MODEM (6809) the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transfered between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also.

X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTAGC-020 users only.

X-TALK Complete (cable, 2 disks)	99.95
X-TALK Software (2 disk only)	$69.95
X-TALK with CMODEM Source Included	$149.95

HIER

A FLEX Hierarchal Disk Directory Program

HIER is a modern hierarchal storage system for users under FLEX. It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size.

Using HIER a regular (any) FLEX disk (8 - 5 - hard disk) can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day solution that all current large systems use.

Each directory looks to FLEX like a regular file, except they have the extension '.DIR'.

A full set of directory handling programs are included, making the operation of HIER simple and straightforward.

A special install package is included to install HIER to your particular version of FLEX. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

* Introduction Special * $69.95

PAT

PAT - A full feature screen oriented TEXT EDITOR with all the best of "PIE (tm)". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX $129.50
* SPECIAL INTRODUCTION OFFER * $79.95
SPECIAL PAT/JUST COMBO (w/source) FLEX $99.95
Note: JUST in "C" source available for OS-9

PROGRAMMERS & USERS TOOLS

SOLVE - OS-9 Levels I and II only. A Symbolic Object/logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 Regular $149.95
* SPECIAL INTRODUCTION OFFER * $69.95

BAS-EDIT

BAS-EDIT - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation.

Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Makes editing BASIC/ XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc.

Complete with over 25 different CRT terminal configuration overlays.

FLEX, CCP, STAR-DOS Regular $69.95
Limited Special Offer: $39.95

CEDRIC

CEDRIC - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

Regular $129.95
* SPECIAL INTRODUCTION OFFER * FLEX $69.95

PASC

PASC - A Flex9 Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package comes complete with source (written in PASC) and documentation.

FLEX $95.00

SPECIAL

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Madia.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

Telex 5106006630

(615)842-4600

SOUTH EAST MADIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

K-BASIC now makes the multitude of TSC XBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (OS-9 or FLEX), including the Assembler

!!! Special !!! ~~$189.00~~ $ 99.49

SAVE $100.00

SCULPTOR

Features

THE SCULPTOR SYSTEM

Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER

Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE

Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY

Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and – without any alterations to the programs – run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

SPEED AND EFFICIENCY

Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

INTERNATIONALLY ACCEPTED

By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe – Sculptor is already at work in over 20 countries.

THE PACKAGE

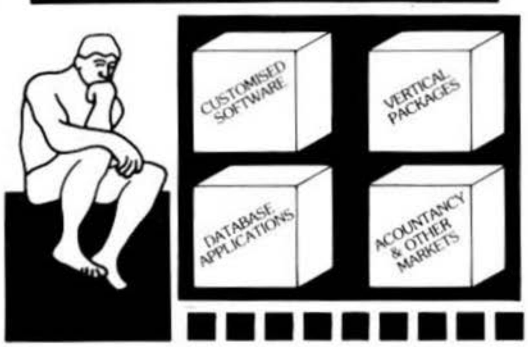
With every development system you receive:

- A manual that makes sense
- A periodic newsletter
- Screen form language
- Report generator
- Menu system
- Query facility
- Set of utility programs
- Sample programs

For resale products, the run-time system is available at a nominal cost.

FIND OUT MORE

SOUTH EAST MADIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE



Facts

DATA DICTIONARY

Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:

- alphanumeric
- integer
- floating point
- money
- date

DATA FILE STRUCTURE

- Packed, fixed length records
- Money stored in lowest currency unit
- Dates stored as integer day numbers

INDEXING TECHNIQUE

Sculptor maintains a B tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one other file.

INPUT DATA VALIDATION

Input data may be validated at three levels:

- automatic by field type
- validation list in data dictionary
- programmer coded logic

ARITHMETIC OPERATORS

-	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
-	Subtraction

RELATIONAL OPERATORS

=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
and	Logical and
or	Logical or
ct	Contains
bw	Begins with

SPECIAL FEATURES

- Full date arithmetic
- Echo suppression for passwords
- Terminal and printer independence
- Parameter passing to sub-programs
- User definable date format

MAXIMA AND MINIMA

Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS

- Define record layout
- Create new indexed file
- Generate standard screen-form program
- Generate standard report program
- Compile screen form program
- Compile report program
- Screen form program interpreter
- Report program interpreter
- Menu interpreter
- Query facility
- Reformat file
- Check file integrity
- Rebuild index
- Alter language and date format
- Setup terminal characteristics
- Setup printer characteristics

SCREEN-FORM LANGUAGE

- Programmer defined options and logic
- Multiple files open in one program
- Default or programmer processing of exception conditions
- Powerful verbs for input, display and file access
- Simultaneous display of multiple records
- Facility to call sub-programs and operating system commands
- Conditional statements
- Subroutines
- Independent of terminal type

Sculptor for 68020
OS-9 and UniFLEX
$995

OS-9 / UniFLEX --		
IBM PC Zenix --	$995.00/$175.00	
MS DOS Network --	*	**
68000 UniFLEX --		
Altos Zenix --	$1595.00/$265.00	
UNIX --	*	**
MS DOS --		
	-- $595.00/$115.00	
PC DOS --	*	**

* Full Developement Package ** Run Time Package Only Full OEM and Dealer Discounts Available!

Telex 5106006630

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the **TSC Macro Assembler**. F, CCF - $99.95

Macro Assembler for TSC -- The FLEX **STANDARD** Assembler.
Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 $99.00

Relocating Assembler w/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers. F,CCF $150.00

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - $98.00

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- (REAL ASSEMBLERS, NOT MACRO SETS) Specify for 180x, 6502, 6801, 6804, 6805, 6809, Z8, Z80, 8048, 8051, 8085, 68000 modular, free-standing cross-assemblers in C, with load/unload utilities and macros.
FLEX, CCF, OS-9, UniFLEX:
Each $50.00 or Any 3 for $100.00 or ALL $200.00
8 bit (not 68000) sources for additional:
Each $50.00 or Any 3 for $100.00 or ALL $300.00

XASM Cross Assemblers for FLEX from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - $150.00

CRASMB from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- $399.00

CRASMB 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000**.
FLEX, CCF, OS-9 $249.00

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCO, Obj. Only $50.00	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 version.

CCF, Obj. Only $100.00	CCO, Obj. Only $ 59.95
F, " " $100.00	O, " " $150.00
	U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. **500+** page Manual with tutorial guide. F, CCF - $198.00

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers**. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, **6-digit Real Numbers**, unlimited sized Arrays (vectors only); interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. N really produces 10% less code than PL/9. F and CCF - $195.00

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - $295.00

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
FLEX, CCF, OS-9 (Level II **ONLY**), U - $575.00

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - $99.95 F 8" $99.95

PASCAL Compiler from **OmegaSoft** (now **Certified Software**) -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.
F and CCF - $425.00 One Year Maint. - $100.00

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now **Fully TSC XBASIC** compatible. The compiler compiles to Assembly Language S urce Code. A NEW, **streamlined**, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Pr grams. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler **with** Assembler - $199.00

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level I COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- $99.95

FORTH from **Stearns Electronics** -- A CoCo **FORTH** Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer **ONLY** - $58.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends --
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

Basic09 XRef from **Southeast Media** -- This **Basic09 Cross Reference** Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.
O & CCO obj. only -- **$39.95**; w/ Source - **$79.95**

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.
F, CCF — **EACH Utility** 5" - **$40.00**, 8" - **$50.00**

DUB from **Southeast Media** -- A UniFLEX "basic" De-Compiler. Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - $219.95

FULL **SCREEN FORMS DISPLAY** from **Computer Systems Consultants** -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF, U - **$25.00**, w/ Source - **$50.00**

DISK UTILITIES

OS-9 VDisk from **Southeast Media** -- For **Level I** only. Use the Extended **Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - **$79.95**; w/ Source - **$149.95**

O-F from **Southeast Media** -- Written in BASIC09 (with Source), includes: **REFORMAT**, a BASIC09 Program that reformats a chosen amount of an **OS-9** disk to **FLEX** Format so it can be used normally by FLEX; and FLEX, a BASIC09 Program that does the actual **read** or **write** function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. **SPECIAL 60 DAY OFFER** **O-$39.95**

COPYMULT from **Southeast Media** -- Copy **LARGE** Disks to several **smaller** disks. FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") **$99.50**

COPYCAT from **Lucidata** -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.
F and CCF 5" - **$50.00** F 8" - **$65.00**

Telex 5106006630
(615)842-4600
SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

FLEX DISK UTILITIES from **Computer Systems Consultants** -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test Disk** for errors; **Compare** two **Disks**; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Resequencer** with **EXTRAs** over "**RENUM**" like check for **missing** label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare**, **Merge**, or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare **two unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC **Cross-Reference** Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC**, **XBASIC**, and **PRECOMPILER** BASIC Programs.
ALL Utilities **include Source** (either BASIC or A.L. Source Code).
F and CCF - **$50.00**
BASIC Utilities ONLY for **UniFLEX** — **$30.00**

COMMUNICATIONS

CMODEM Telecommunications Program from **Computer Systems Consultants, Inc.** -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX: with complete Source - **$100.00**
without Source - **$50.00**

XDATA from **Southeast Media** -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - **$299.99**

GAME

RAPIER - 6809 **Chess** Program from **Southeast Media** -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - **$79.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

SOUTH EAST MEDIA
5900 Cassandra Smith Rd
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —
F = FLEX, **CCF** = Color Computer FLEX
O = OS-9, **CCO** = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

Telex 5106006630

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343

for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

BUSINESS

WORD PROCESSING

SCREDITOR III from **Windrush Micro Systems** -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - **$175.00**

STYLO-GRAPH from **Great Plains Computer Co.** -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES —> CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from **Great Plains Computer Co.** -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES —> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from **Great Plains Computer Co.** -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES —> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from **Southeast Media** -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the **Display**. Use the **PPRINT.CMD** supplied for producing multiple copies of the "Formatted" Text on the Printer **INCLUDING IMBEDDED PRINTER COMMANDS** (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: FLEX - CC
Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent **BUDGET PRICED** word processor, **with features none of the others have.** This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - **$49.95**
Disk Set (2) - F & CCF & OS9 (C version) - **$69.95**

SPELLB "Computer Dictionary" from **Southeast Media** -- **OVER 150,000 words!** Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

!! SPECIAL LIMITED TIME OFFER !! F and CCF - $99.95

DATA BASE - ACCOUNTING

XDMS from **Westchester Applied Business Systems** -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for Giant Data Bases. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc. XDMS Level IV see Westchester Applied Business Systems ad this issue.

XDMS System Manual - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95
Upgrades to Lvl IV - $250.00 XDMS Lvl IV - F & CCF - $350.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from **Computer Systems Consultants** -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires **TSC's Extended BASIC.**
F and CCF, U - **$50.00**, w/ Source - **$100.00**

DYNACALC from **Computer Systems Center** -- Electronic Spread Sheet for the 6809.
F, **SPECIAL** CCF and OS9 - **$200.00**, U - **$395.00**

FULL SCREEN INVENTORY/MRP from **Computer Systems Consultants** -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires **TSC's Extended BASIC.**
F and CCF, U - **$50.00**, w/ Source - **$100.00**

FULL SCREEN MAILING LIST from **Computer Systems Consultants** -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires **TSC's Extended BASIC.**
F and CCF, U - **$50.00**, w/ Source - **$100.00**

DIET-TRAC Forecaster from **Southeast Media** -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - **$59.95**, U - **$89.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
**OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

horizontal reference signal is compared to the H sync signal from the RMI and thus forms a PLL locking the 36 Mhz oscillator such that the horizontal sync has the correct phase relationship. PAL identification is obtained from the subcarrier lock system (indentification detector, PAL Flip Flop). The internally generated PAL chroma is then controlled from this identification phase. In the NTSC mode, the identification system is disabled.

Composite Video Generation

The red, green and blue (RGB) analog signals from the RMC are used to generate composite color NTSC or PAL signals. First, color difference and luminance (Y) signals are generated by resistively matrixing RGB. The color difference outputs of the resistive matrix (R-Y and B-Y) then drive two double balanced chroma modulators. The carriers for the modulators are supplied in quadrature from the divide by four ring counter output of the crystal oscillator. The modulator outputs are combined to form the chroma signal and are added to the Y signal along with composite sync to form the complete composite color video signal. In the PAL mode, the R-Y color difference signal changes polarity on alternate horizontal lines.

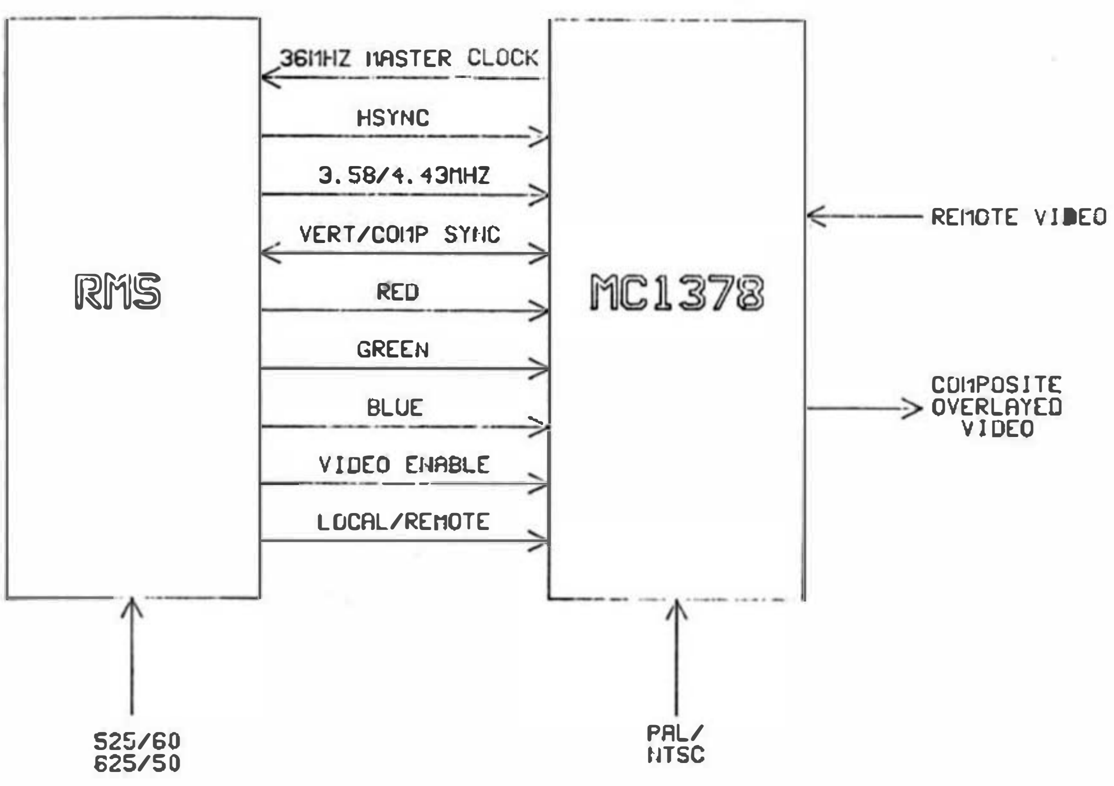
In the remote mode, the phase and the amplitude of the chroma signal is matched to that of the external video signal so that the two composite color signals are completely compatible and may be displayed in an overlay format by operating the video enable change over switch.

Building a RMS based graphics system

A general RMS system based on a MC68000 microprocessor is shown in figure 5. In such a system, the RMS is assigned 1 megabyte of the 16 megabyte address space by decoding the upper four address lines to form a chip select for the RMS. In this basic configuration, the remaining 19 lines, plus Upper Data Strobe, are connected to the system X bus through three 74ALS257 multiplexers. The 20 address lines are time division multiplexed along with RMS memory accesses using only 10 pins.

The data bus is interfaced to Port A and Port B of the RMC using two packages of 74ALS244 bus drivers and two packages of 74ALS374 octal latches. The dynamic RAM is organized in byte-wide banks. Banks 0 and 1 are used together to address 16-bit words and separately to address bytes. Banks 2 and 3 are used similiarly. Banks 1 and 3 are always the least significant byte of a 16-bit word and are connected to Port A, while the most significant byte, Banks 0 and 2, are connected to Port B.

The system works with a 8 Mhz version microprocessor for which the RMI provides both the clock (7.95 Mhz) and the asychronous handshake, DTACK. In higher performance applications, the designer may supply a separate higher frequency clock to the MPU , however, access to the RMS will always occur at the RMS rates.



The video interface block shown in figure 5 can be one of several types of interfaces. The simplest interface is to buffer the RGB and composite sync then drive a monitor directly. In other applications, the choice might be to create a composite video source, using the MC1377, RGB encoder, or in overley work, use the MC1378, overlay synchronizer. Additionally, separate horizontal and vertical syncs can be derived, as required by some monitors, providing a variety of ways to interface the output video.

This basic system configuration can be expanded or compressed with minimal hardware impact from application to application. It has numerous system variables defined for the system designer,yet allows enough freedom for a custom design. The basic 68000 system can be implemented in a 68008 system for cost effective applications without sacrificing software efforts. Minimum hardware applications can use the MC6809E microprocessor with only one bank of memory. Again, the RMS will configure the correct timing for that MPU, while the hardware impact consist of changing the MPU interface logic in a predefined configuration.

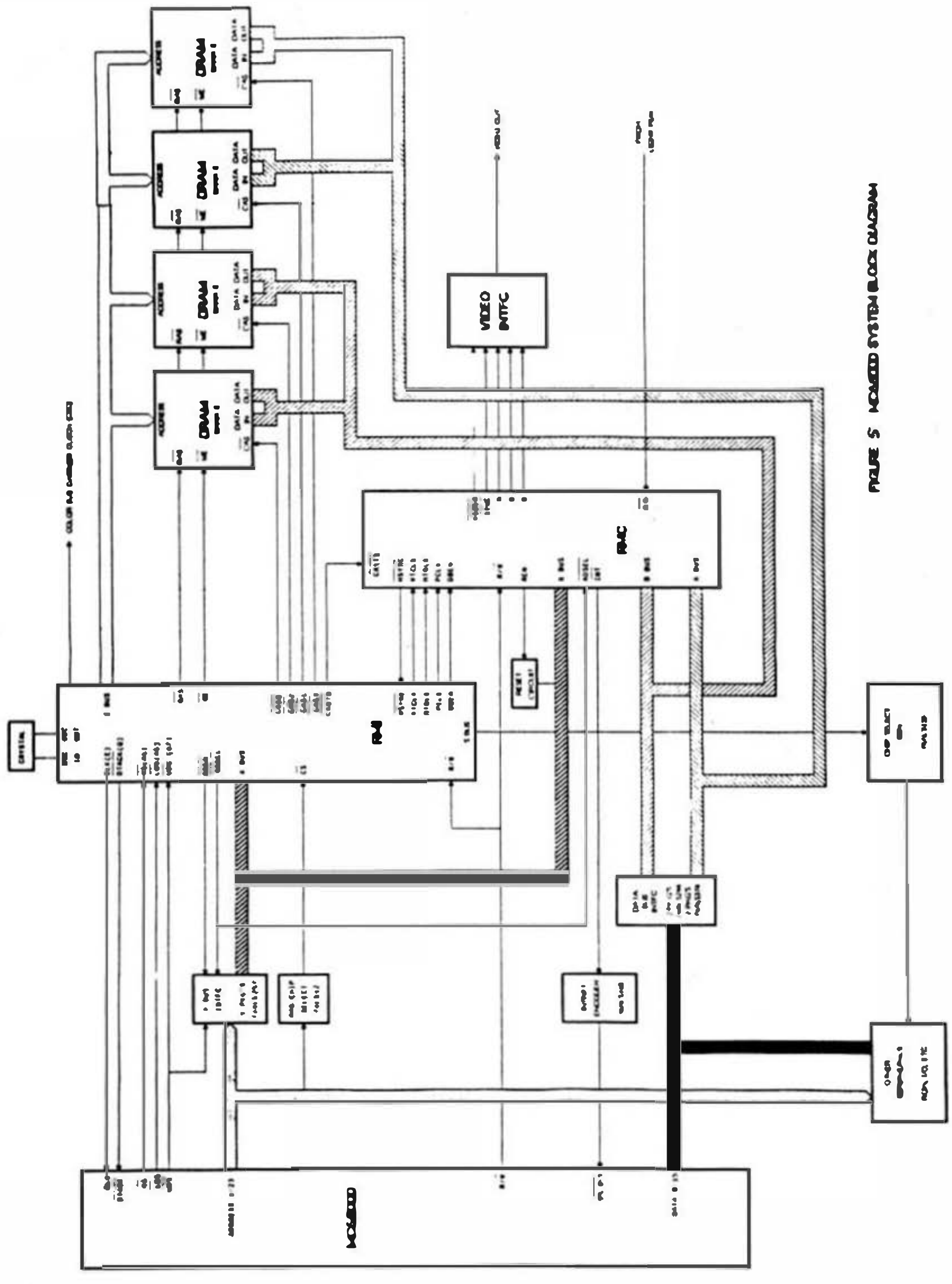


FIGURE 5 MC68000 SYSTEM BLOCK DIAGRAM

QPL

QPL - A Bold Step to Very High Level Language

This is Part 2 of a series of articles about the QPL programming language. QPL is available for Flex-9 systems from Compiler Products Unlimited, Inc. Phone (602) 991 1657

In the first article, we covered many of the 'vanilla' features of QPL; the arithmetic, the compare-branch-&-loop, variable names, and input and output.

In Part 2, we will discuss the pattern matching features of QPL. In the first section of this installment, we will consider the various types of patterns, including alternation, conditional assignment, and Special Patterns. In the second part of this installment, we will look at examples of pattern matching.

PATTERN MATCHING:

What are the applications for pattern matching? In QPL, we use pattern matching to do the following types of tasks:

1. Determine if a string starts with a given pattern of characters.
2. Determine if a string starts with any of several alternate patterns of characters.
3. Determine if a string starts with a given pattern of characters, followed by another pattern of characters.
4. Determine if a string contains a given pattern of characters, anywhere, not just at the start.
5. Extract sub-strings from a larger string.
6. Extract data fields from file records.

In QPL, pattern matching is done by the built-in function MATCH, which takes two arguments; a string, and a pattern. For example, we define STRING1 = "This is a string". To determine if STRING1 starts with 'Th', we could write MATCH(STRING1,"Th"). The MATCH function sets the Success/Fail flag which can cause conditional branching as described in Part 1. In the example above, MATCH would succeed. Note that in this pattern match, we have determined that STRING1 starts with 'Th', but we do not care what follows. It could be 'This' or 'That'.

ALTERNATES:

It is very common to need to know if a string starts with any of several alternates, for example when interpreting command input in a man-machine-interface situation. To do this kind of pattern- matching, we must create a pattern which contains not only text, but instructions to the MATCH function. Suppose we want to recognize color commands. We would create a pattern of color alternates by the following line:

COLORS = "RED" | "YELLOW" | "GREEN" | U1 | U2 | U3

The pattern COLORS contains not only the values but the

It is very common to need to know if a string starts with any of several alternates, for example when interpreting command input in a man-machine-interface situation.

alternation instruction '|' which is read as 'or'. Now we have a pattern which can be used to recognize any of six colors. The first three it can recognize are 'hard coded' as quoted text literals RED, YELLOW, GREEN. The forth through sixth colors it can recognize are whatever U1, U2, and U3 contain. For example, we can make U1 = "BLUE". To use the pattern COLORS in a MATCH statement with STRING1, we would write:

MATCH(STRING1,COLORS). As before, if STRING1 started with any of the colors contained in the pattern COLORS, the match will succeed. Now suppose we wanted to know which color in the pattern COLORS matched STRING1. There are two ways of getting this information. One way is by the value returned from match.

VALUE RETURNED by MATCH

The MATCH function returns a value which is the number of the alternate which matched. The first alternate is 0. To use the value returned by MATCH, we would rewrite the MATCH statement above as WHICH_ONE = MATCH(STRING1,COLORS). When we say MATCH returns a value, what it means is that the match statement is replaced by the value it returns. So if STRING1 contains "GREEN", MATCH will return 2, and the MATCH(STRING1,COLORS) part of the statement will be replaced by 2, so that it now reads WHICH_ONE = 2. Obviously, after the complete statement executes, WHICH_ONE will have the value of 2. This tells us which alternate matched.

The other way to determine which alternate matched is by using conditional assignment.

CONDITIONAL ASSIGNMENT:

Conditional assignment means "If this pattern matches, copy what it matched to a variable". In order to perform conditional assignment, we use the conditional-assignment operator, which is a period '.' or dot. To use conditional assignment with the pattern COLORS, we would rewrite COLORS as follows:

COLORS = "RED" | "YELLOW" | "GREEN" | U1 | U2 | U3 . FOUND

Now when we use the pattern COLORS in a MATCH statement, if any of the alternate colors match the value of the string, that part of STRING1 which matches will be assigned to FOUND. If there is no match, the value of FOUND will be unchanged. It is important to note that only that part of STRING1 which matches one of the alternates of COLOR will be copied to FOUND. For

instance, if STRING1 = "YELLOWBIRD", when we perform the MATCH(STRING1,COLORS) then FOUND will contain 'YELLOW'.

SPLITTING STRINGS:

We sometimes want to split strings for text analysis. For example, suppose STRING1 may contain either 'YELLOWBIRDS' or 'YELLOWFISH' or 'BLUEBIRDS' or 'BLUEFISH'. We want to determine what color we have, and weather they are fish or fowl. We could do this by making a pattern:

```
ANALYZE = ("YELLOW" | "BLUE") . COLOR & ("BIRDS" | "FISH") . ANIMAL
Now, suppose STRING1 = "BLUEBIRDS".

MATCH(STRING1,ANALYZE) will proceed as follows:
 Try 'YELLOW'  - no match there
 Try 'BLUE'    - that matches, assign 'BLUE' to COLOR.
 Resume match scan from end of 'BLUE'
 Try 'BIRDS'   - that matches, assign 'BIRDS' to ANIMAL
 Set success flag and return 0 (the first alternate)
```

In the above example, we split a string based on detailed information about what the string contained. It is often more useful to be able to perform matching when we have less detailed information. To do this, we use the Special Patterns.

SPECIAL PATTERNS:

The Special Patterns are patterns produced by pattern generator functions. These functions are SPAN, BREAK, LEN, SPLIT, ANY, and NOTANY.

LEN:

To use the LEN pattern-generator function we would write a statement like: SP20 = LEN(20). This would generate a special pattern and assign it to SP20. The patterns that LEN generate match any characters up to the number specified in the LEN argument. Thus SP20 will contain a pattern which matches a run of 20 characters, regardless of what they are. LEN always succeeds (sets success/fail flag to success), except when the string it is matched against is of zero length. LEN patterns with conditional assignment attached are very useful for string splitting. For example,

SP20 = LEN(20) . FIRST

creates a pattern which matches a run of 20 characters and assigns them to FIRST.

BREAK:

The BREAK pattern constructor function produces a pattern which will match a run of characters up to (but not including) any character in the BREAK argument. For example,

PBRK = BREAK(".,;:")

produces a pattern which matches a string of characters up to any one of the characters '.,;:'

SPLIT:

Sometimes we want to make a pattern which acts like a BREAK pattern, but we want it to stop not on any single character in the argument, but on a sequence of characters. To do

SPLIT_PAT = SPLIT("$A") will create a pattern which will match all characters until it finds the exact string '$A'. We could use conditional assignment to copy everything the SPLIT pattern matches to a variable. During matching, if the SPLIT argument pattern is not in the MATCH string, the match fails (and conditional assignment is not done). SPLIT patterns also provide us with a way to determine if a long string contains another string somewhere in it (rather than just at the start).

SPAN:

SPAN produces a pattern which is the logical inversion of BREAK patterns. That is, a SPAN pattern will match a run of characters up until it finds a character which is not in the SPAN argument. SPAN_PAT = ("1234567890") creates a pattern which will match a string of characters up until it finds anything which is not a number character. This pattern of all numbers is useful for determining if a string is a numeric integer.

ANY:

This constructor function produces a pattern which matches only a single character, but it may be any one of several characters in the argument. For example, ANY("1234567890") creates a pattern which will match only 1 character, which must be any one of 1234567890.

NOTANY

This is the logical inversion of ANY. It produces a pattern which matches a single character, which must not be in the argument.

APPLICATION EXAMPLES:

Our first application example will be an adventure game.

The game will read a data file named ST1.TXT then present the player with a 'room description'. It will then ask the player for a 'move' command. Depending on which direction the player decides to move, the code will extract a new room from the data file. The room description for the new room will be read from the data file and presented to the player, and so on.

The data file for this simple demonstration program is shown in figure 1. It consists of three 'room records'.

There may be up to 10 room records in the game data file, with the limit determined by the size of array ROOMS.

In the data file in figure 1, all the text up to the first '$' is room-description, which the program will read from the text file and display on the screen. The sub-string 'NXSX$E1$W2$' encodes these facts: There are no valid 'exits' either north or south. Going east will move you to room 1, going west will move you to room 2. The final '$' line marks the end of a room record. We will require that this is the only thing on the last line, to simplify the program, and also visually group each room record in the file.

A QPL program consists of data, patterns, and code. We have defined the data, and will next build the patterns we will need.

We will need a pattern which will cause all text up to the first '$' to be printed in the screen. This 'display' pattern will be called DESP_PAT. Refering to figure 2, we see that DESP_PAT will 'match' everything in a string up to '$' and will assign all it matches to OUTPUT. Remembering that anything assigned to OUTPUT will be printed, you can see how DESP_PAT will work.

Next, we will build some patterns which will scan the data and make the room transitions for us. These patterns will be called NORTH_PAT, SOUTH_PAT, EAST_PAT,

and WEST_PAT. Again refering to figure 2, we see that each of these patterns consist of a SPLIT special pattern which will scan to the required part of the room-transition. Following the SPLIT is a concatenated LEN(2) which will scan over two characters (the '$E', etc) in the room transition. Next is a BREAK("$") which will match all following characters up to the next '$'. Last is a conditional assignment which assigns the last concatenated match (the BREAK("$")) to NEW_ROOM. Note that conditional assignment is associated to the left concatenated pattern in a compound pattern such as NORTH_PAT. Now with these direction_PAT patterns we have a set of tools which will extract the next-room in a manner which is independent of the number of characters in the room number.

Note that our data file contains delimiter characters ('$') but that it is human-readable file. Also note that no special formatting was required; we did not need leading 0's in the room numbers like 002. The text file can be prepared using an editor, and minimal instructions to the person 'writing' the adventure game. Also note that the game itself is completely described by a text file which is external to the game program. In this way we have partitioned the task into two independent sub-tasks, which can be worked on by two people, with little interaction. QPL makes it easy to partition program design tasks because all interactions between QPL programs are in the form of text. Debug is also simplified because people can easily read QPL files.

The next pattern we will need is a command-recognition pattern, called CMD_PAT. Figure 2 shows that CMD_PAT is just a 5-string alternation pattern.

We will design the program in the following manner:

When the adventure program is executed, it will open the data file ST1.TXT and will read each room record into an array. In this way we will have all room '0' data in array element 0, and so on. There is no need for concerns like "will the room record fit in the array element (too many characters?)". Everything always fits in QPL array elements! In reading from the file, QPL always reads an entire line, terminated by a carriage return.

The simple 'adventure' game of figure 2 is coded for clarity, rather then for small code size. You may notice that the move- patterns NORTH_PAT, SOUTH_PAT, etc., differ by only 1 character, the N,S,E or W. We could write the move-patterns as one generic pattern by replacing the literal "$N", "$S", etc, with a variable named DIR. The variable DIR would then contain the command that was input (N,S,E,W) with a '$' prepended using concatenation. This change would allow us to change the MATCH statements at the labels DO_N, DO_S, etc. into a single statement. We would save three lines of pattern definition code, and 3 lines of MATCH code, not counting the labels.

This simple program can be upgraded to a full feature general purpose adventure game program with the addition of about another 70 lines of code. The full feature program utilizes virtual memory techniques to allow the game to be as big as 1000 rooms, with only a 10 element ROOMS array. We will release the full feature program in a about 1 month.

SUMMARY:

In this simple 'adventure' program we have seen that QPL uses data structures which easily accommodate varying types and sizes of data, such as occur in real-world programs. QPL file records are plain text files, which makes for easy communication between QPL programs and programs written in other languages. Extracting data from mixed-data records is easy with patterns containing conditional assignment. These QPL features form a programming language which gets results fast, and produces programs which are easy to modify.

In the next QPL report, we will look at user-defined functions, and some functions which allow us to build large programs from smaller program segments.

```
* *****   FIGURE 1 - DATA FILE 'ST1.TXT'  ******
You are standing in a clearing in a woods.  To the east is a small
cabin.  To the west is a rocky ravine.  North and south lead into
dense weedy brush.  $NX$SX$E1$W2$
$
This is room 1.  $NX$SX$E2$W0$
$
This is room 2  $NX$SX$EX$W1$
$
* *****  END  FIGURE 1 - DATA FILE 'ST1.TXT'  *****
```

```
* *****  FIGURE 2 - ADVENTURE PROGRAM  *****
  ARRAY(ROOMS,10)
* First, we build the required patterns.
  DESP_PAT = BREAK("$") . OUTPUT
  NORTH_PAT = SPLIT("$N") & LEN(2) & BREAK("$") . NEW_ROOM
  SOUTH_PAT = SPLIT("$S") & LEN(2) & BREAK("$") . NEW_ROOM
  EAST_PAT = SPLIT("$E") & LEN(2) & BREAK("$") . NEW_ROOM
  WEST_PAT = SPLIT("$W") & LEN(2) & BREAK("$") . NEW_ROOM
  CMD_PAT = "N" | "S" | "E" | "W" | "Q"

* Next, we will read all the room data into the array ROOMS.
  OPEN(1,"ST1.TXT",GET,SEQUENTIAL)
  RM_NUM = 0
  DATA = NULL
READ_LOOP
  STATUS = READ(ST1.TXT,STUFF)
* Check to see if we have read to file end.
  EQ(STATUS,0)      :F(FILE_END)
* Check to see if the line read is the end of the room record.
  IDENT(STUFF,"$")      :S(HAVE_ROOM)
  DATA = DATA & STUFF    :(READ_LOOP)
HAVE_ROOM
* Put the room record into the array ROOMS.
  ROOMS<RM_NUM> = DATA
  RM_NUM = RM_NUM + 1
  DATA = NULL                  :(READ_LOOP)


* Next, we code the main loop of this simple program

FILE_END        CLOSE(ST1.TXT)
  RM_NUM = 0
MAIN_LOOP
  TEXT = ROOMS<RM_NUM>
  MATCH(TEXT,DESP_PAT)
CMD_LOOP
  OUTPUT = "What is your command?"
  CMD = INPUT
  MATCH(CMD,CMD_PAT)       :S(XQT_CMD)
  OUTPUT = "I don't understand that command"  :(MAIN_LOOP)


XQT_CMD
* Here we prepend 'DO_' to the move command, so we can do a
* goto-indirect on the value of CMD.  '$' is indirection.
  CMD = "DO_" & CMD      :($CMD)
DO_N
  MATCH(TEXT,NORTH_PAT)    :(GO1)
DO_S
  MATCH(TEXT,SOUTH_PAT)    :(GO1)
DO_E
  MATCH(TEXT,EAST_PAT)     :(GO1)
DO_W
  MATCH(TEXT,WEST_PAT)

GO1
  IDENT(NEW_ROOM,"X")    :S(NO_EXIT)
  RM_NUM = NEW_ROOM    :(MAIN_LOOP)

NO_EXIT  OUTPUT = " There is no exit in that direction"  :(CMD_LOOP)

DO_Q
* This is the program end, and the 'QUIT' exit.

* *****  END FIGURE 2 - ADVENTURE PROGRAM  *****
```

Bit Bucket

Modem68 Updated

John Moorfoot

Copyrighted 1986 - Computer Publishing, Inc. (CPI) and 68 Micro Journal.

DISTRIBUTION RIGHTS:

FIRST RELEASE:
Authors copyright notice: I allow unrestricted distribution of this software on a non-profit basis. However any commercial application using either the programs or documentation requires my written permission in advance.

SECOND RELEASE:
Publishers copyright notice: *** Computer Publishing Inc. also places additional copyright to this software. To wit: it is specificially intended that the software published herein, and recorded to a 68 Micro Journal "Reader Service Disk", ie strictly for non-profit distribution, as defined by the copyright and trademark rules and regulations as adopted by the most recent action of the International Copyright Convention. Any use in conflict with this stipulation shall be enforced to the full extent of the law!

Also it is stipulated that ANY first or second source distributor shall not distribute more than 5 copies of these aforementioned on a non-profit basis, within any 12 month period. Non-profit distribution of whole or partial parts of these works, in amounts in excess of five (5) whole or partial parts of these works shall require written exception from both parties indicated above as holders of copyright titles. ***

Releases prior to publication date remain in existence.

NOTE: Both published copyright notices above (authors and Publishers) must be complied with! These restrictions are imposed due to illegal reproduction, for profit, having occured, to material published previously. They in no way are intended to inhibit the copying and distribution, by indivuduals, for hobby or non-profit use. Volume distributors must secure signed agreements.

No additional releases authorized.

MODEM68 provides for file transfer between two computers using the FLEX operating system, or between a FLEX based machine and a CP/M computer. FLEX utility commands are supported from within MODEM68. The program uses Ward Christensen's protocol for file transfers with Chuck Forsberg's YAM batch file transfer mode.

The Christensen protocol is widely used in micro- computer applications and has been well documented in Byte and other journals. The only difference for a header block under YAM and a normal Christensen block is that the header has block number 0. The filename is sent as it appears on the screen in CP/M format and may or may not have a period and extension. The filename is terminated by a null. On completion of file transfer(s) a null filename is sent to terminate the session. For FLEX users, the standard FLEX conventions apply - e.g. .ASM will transfer any files with a ASM extension, however due to differencee in filename conventions some CP/M filename characters may be translated into an alpha character for FLEX (lowercase z in this version). Also, any CP/M files received without an extension will default to .TXT.

Files of any size can be handled, as there is no requirement for them to fit into memory. The I/O buffer has been restricted to the Christensen protocol block size of 128 bytes for ease of implementation. In practice this does not significantly increase file transfer times, although it may slow things down in terminal mode with data logging.

USAGE:

The program is menu driven and prompts for all input when required. To return to the main menu, (aborting a file transfer) type a ^E (control E) from the keyboard. All FLEX utilities obey the same conventions they use normally - e.g. COPY 0 1 will copy all files from drive 0 to drive 1. Switches have been included to provide for batch file transmission, a means of logging all screen display to disk using XON/XOFF protocol, for echoplex operation, to allow proper operation between two computers running FLEX, and to reduce file transmission errors by using cyclic redundancy checking. These switches can be viewed by use of the menu item Z, and may be altered if desired. A carriage return entered in response to the prompt will return to the main menu without displaying the remainder of the switch settings.

DESCRIPTION:

The entire program is written in M6800 assembler code to allow for assembly on FLEX2 and FLEX9 computers.

MODEM68

This is a header file which calls in a number of library files - MOD68-1 to MOD68-11, contains configuration parameters, and has the default settings for the soft switches.

MOD68-1

This file contains all equates used. It also contains some buffer space on page 0 for common variables.

MOD68-2

This is the mainline routine. It prints an initial signon message followed by a menu. All other modes (except return to FLEX) are called as subroutines and return to the menu on completion. Responses to the menu prompts are independent of case.

MOD68-3

All subroutines used by more than one module are contained in this file. There are also initialisation routines for ACIA's used as a modem port.

MOD68-4

Resets the stack pointer and returns to FLEX. This routine also calls the file management system to close any files left open.

MOD68-5

This file contains all file reception stuff. It firstly checks to see what mode is used (normal or batch) and then uses the appropriate routine. Normal mode is fairly straightforward, it prompts for a filename, opens the file and then waits until the line is free to send an initial NAK. Batch mode is mostly concerned with the differences between FLEX and CP/M conventions. Once it gets that all sorted out it calls the normal receive routines until all transfers are completed. You may wish to alter the default compatibility character 'z' in the header file, but this should be set to an ALPHA character, as FLEX requires the first character of a filename or extent to be alpha.

MOD68-6

This routine handles terminal communications mode. If in log mode, you will be prompted for a filename to which all screen data will be logged until you return to the main menu. This mode uses XON/XOFF protocol and is useful for capturing data from bulletin boards etc. When the buffer is nearly full, the system sends an XOFF to halt output from the remote computer. It then waits for a bit to see if anything else arrives before it writes the buffer out to disk. After completion of the write, it sends an XON to restart output from the remote computer. At present, the number of characters positions left in the buffer before an XOFF is sent is 10. This seems to work ok with an RCPM running at 300bps but may need adjustment for other systems or speeds. If this is the case, change margin in MODEM68.TXT. If the ECHO switch is set, all input from the keyboard is displayed on screen after being sent to line, and anything received from line is returned. Without echo, only characters received from line are displayed. For dialogue between two systems using MODEM68, one must be in echo mode. Also when communicating with a CP/M computer running MODEM or YAM, the CP/M machine must be in CHAT mode or the FLEX machine must be in echo mode.

MOD68-7

This file contains the file transmission bits. Similarly to receipt, the mode is checked first. Much of MOD68-5 (receive file) and MOD68-7 (transmit file) are concerned with the differences between FLEX and CP/M conventions. FLEX allows alpha characters only as the first character in a filename and an extension. It also requires an extension, so the additional characters allowed by CP/M are forced to an alpha character when in batch mode. No attempt has been made to implement file transmission times because FLEX uses space compression, so the amount of data stored is less than that transmitted. The batch send routine uses FLEX filename matching conventions. The code is based on the TSC pdel utility, and performs similarly for name matching, however the filename is accepted from within the program instead of appearing on the command line.

MOD68-8

This routine allows any utility running in the utility command space within FLEX to be run. It issues the normal FLEX prompt when called, and appears to the user as if he is in FLEX. Any program that runs in low memory must not be executed with this routine - e.g. EDIT. When the utility has completed, it will return to the MODEM68 main menu.

MOD68-9

This routine can be used to abort file transfers from a remote computer.

MOD68-10

This is a routine to display and/or alter switches. Any additional switches should appear in MOD68-1 before endsw and should have a five letter name (including spaces) included in swmsgf in MOD68-11.

MOD68-11

This file contains the command table, all strings and messages and file control blocks and buffers.

MOD68-12

This module contains routines to dynamically reconfigure a 6551 ACIA used as a modem port. You can alter baud rates, number of data bits, number of stop bits, parity, state of RTS, and send a break. Similar functions could be written for a 6850, but baud rate changes are limited.

MOD68-13

Toggles XMODEM mode on/off. This allows you to control a computer running MODEM68 from a remote terminal connected to the modem port. (This could possibly adapted to a bulletin board system).

MOD68-14

Lists HELP.M68. NOTE: LIST.CMD must be on the system drive, and HELP.M68 must be on the working drive.

ADAPTING TO YOUR SYSTEM:

If you are using a 6850 ACIA for keyboard input and either a 6850 or 6551 ACIA for your modem port, the only changes required will be in the header file MODEM68. Requirements for other systems are discussed later.

One of the operating system equates must be set true, the other false. Set for your system e.g. for FLEX9 set FLEX9 equ true FLEX2 equ false. Similarly set one of a6551 and a6850 true and the other false to reflect your modem chip.

The default switch settings may be changed to suit your requirements. Change to fcb $ff to turn a switch on. For a description of the switch actions list HELP.M68.

The base address for your modem port and keyboard port must also be set. Also, modem port default parameters should be inserted. For a 6850, acl is a master reset and should not need to be changed, ac2 is the control word. For a 6551, acm is the command register and should not normally need changing, acl is the control register and should be set for default baud rate requirements. Some values for acl are: $16 - 300bps, $1e - 9600bps, $0 - external clock.

A single character to home the cursor and clear the screen is set as ff equ $c which is a form feed. If your terminal requires more than one character or does not support this feature, it can be set to $0. This sequence is only used to start the menus at the top of the screen.

The equate to prevent lose of data when logging screen output to a disk file is currently set to 10 in margin. This is the number of characters which can be accepted after MODEM68 sends an XOFF to halt the sending end. If you find you are losing data in LOG mode, increase this value.

As filename conventions differ from CP/M MSDOS/PCDOS to FLEX, a lowercase z is substituted in BATCH mode whenever an illegal FLEX character is discovered. To change to another character, replace the z in the compat equate in the header with another ALPHA character. This is because FLEX only allows alpha characters as the start of a filename and extension,

Users who have other than a 6850 for keyboard input will need to supply there own routine for keyboard status in MOD68-3.T13. FLEX9 users may be able to use the following:

```
kbdat jsr $cd4e (STAT FLEX call)
 tfr cc,b
 ror b
 ror b
 ror b
 rts
```

The keyboard input routine will also need to be changed, but

```
kbdin jsr getch
 rts
```

should suffice without degrading performance significantly.

Users with devices other than 6850 or 6551 for their modem port will need to write routines for input status, output status, modem input, modem output and port initialisation. The requirements for these routines can be found in MOD68-3.T13.

CHANGES FROM VERSION 1.0 to 1.0.4

All user changeable parameters are now contained in the header file MODEM68.TXT. Support for 6551 ACIA's has been added, as has XMODEM mode and cyclic redundancy checking. (The crc algorithm was transcribed from a CP/M module, and I have no idea how it works except that it implements a polynomial - I leave it to those far more mathematically competent to explain). FLEX mode has been added to correctly handle binary file transmission between FLEX computers, and at the same time, inconsistencies in the filename character substitution have been corrected. I have changed the extension of the library files to allow me to quickly locate different versions of a module.

The command loop (MOD68-2) has been changed to alter the prompt for XMODEM mode so that you can tell which end is which when operating remote.

6551 initialisation, XMODEM input and output and cyclic redundancy check routines have been added to MOD68-3, and purge has been changed for crc mode. Also, the error routine has been changed to close any files left open when an error occurs. A new routine has been added to check for binary files and to set space compression off if found.

MOD68-4 now includes a call to reset I/O vectors which may have been modified by XMODEM.

MOD68-5 has code added to check for binary files. A problem with timeouts in non-batch mode has been corrected, and crc checking has been included.

In MOD68-6 an alternate return to main menu character ^X has been added to terminal mode to allow a user to escape from terminal mode in XMODEM.

Binary file handling and crc checking have been included in MOD68-7.

Additional code in MOD68-8 is to prevent changing XMODEM mode after executing a FLEX utility.

MOD68-11 has additional tables and messages to support the new routines.

IN CASE OF TROUBLE:

I welcome any suggestions or bug reports. Also any enhancements (such as CRC checking) are encouraged.

N.B. FLEX is a trademark of Technical Systems Consultants. CP/M is a trademark of Digital Research, MSDOS is a trademark of Microsoft, and PCDOS is a trademark of IBM.

*The updated version of MODEM68 is now 68' Micro Journal Reader Service Disk #12. See page 62 this issue for details on how to order.

PASC Review

Art Weller

PASC is the product of:

Mr. Hugh Anderson
109 Aro St.
Wellington, New Zealand

As implied by its name, PASC is a Flex9 compiler with a definite Pascal "flavor", and anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. A page in the manual explains the differences succinctly:

"PASC has a block structure similar to PASCAL, although PROCEDURES may not be nested.

PASC doesn't have

- TYPES,RECORDS,SETS,FILES
- Multi-dimensional arrays
- Array lower bound specifications
- More than 2 lexical levels
- Assignment of structures larger than 16bits
- GOTOs
- REALs
- A FOR - loop
- The WITH statement
- IO - i.e. the WRITE or READ statements

PASC does have

- Easy (UNCONTROLLED) access to memory addresses
- INTERRUPT procedures
- Constant Arrays (The DATA declaration)
- A CODE statement to allow inline insertion of machine code
- ORG and STACK directives

In addition, pointers and strings are different, and call-by-reference is done by hand."

This list of reserved words (keywords) will give a better indication of the scope of this compiler:

AND	ARRAY	BEGIN	BYTE
CASE	CODE	CONST	DATA
DO	ELSE	END	EXTERNAL
FORWARD	FUNCTION	IF	INCLUDE
INTEGER	INTERRUPT	MODULE	NOT
OF	OR	OTHERWISE	PROGRAM
PROCEDURE	REPEAT	RETURN	THEN
UNTIL	VAR	WHILE	XOR

The otherwise excellent manual does not discuss about 1/4 of these, but presumes that the user has had prior experience with Pascal. In a way, that's a shame, for PASC would provide an excellent introductory compiler for a first-time user. It should also be noted that PASC is an "integer-only" implementation, a limitation not in keeping with the rest of its capabilities. The addition of REALs would greatly enhance its utility.

The good news is that PASC departs from the usual practice of providing a run-time-interpreter for the compiled output and instead generates ROMable, relocatable, reentrant native (6809) object code; far more compact and significantly faster. (As Hugh Anderson says, "Who would bother writing any other sort (of compiler) for the 6809?"). In fact, the "run-time package" consists only of the tiny integer arithmetic. This aspect and the fact that there is provision for easy access to RAM addresses and programmer control over ORG and STACK makes it a good choice for the design of utilities, system software, or ROMable code. No command line options are provided, but compiler directives may be included in the program file to control the type of output (source listing, assembler output, binary, and a "debugging" feature). The assembler output would, of course, allow optimizing some critical portion of the program where the use of CODE is inappropriate.

PASC uses rigid syntax and structure (as does Pascal). Careless typists like me usually have a bit of debugging to do on new programs because of syntax errors while concentrating on the flow of the program. PASC is unforgiving of these errors, stopping dead in its tracks on each one. This isn't quite as bad as it sounds, since many of the compilers that list out all the errors in one run actually get "sick" after the first one or two so that the later ones become meaningless.

In hope of taking some of the pain out of this, the PASC set includes a syntax/structure editor called ED. It makes no pretext about being useful as a text editor, and cannot be, since it is based on a table of the language keywords and attributes. The purpose is to guide the programmer in producing perfect source code (in the syntax/structure sense, that is) -- ie, the use of ED in entering the program will result in no compile time errors due to these causes.

I found the concept and implementation of ED fascinating; to the point of spending more time learning ED than PASC. And it does, in fact, constrain the programmer to the requirements of the language. A detailed explanation would take too much space here, but some idea can be gained by visualizing a tree-like arrangement representing syntax required in the use of each keyword. ED allows (requires) tabbing through the tree-structure to find the particular "branch" that is needed, and at each of these points (nodes) it is possible to expand or contract the display to examine the options available at that point. Note the use of the word "constrain" above. While the basic idea seemed a spendid one, the way ED interfaces with its user is tedious, to say the least. The mechanics of using ED are so distracting that I found myself concentrating on use of the editor rather than the program I was trying to work on. However, I can see a great deal of potential in this technique and hope further development effort goes into improving it, since it could be used as a programming tool for other languages as well. In fact, ED came with a file to convert to Pascal use.

The review disk came with a couple of "freebee" programs (not copy-righted, that is) that should be mentioned. The CHESS source would give some good examples of the use of PASC, but being incomplete, is not of much use in actually playing the game.

There is also a screen oriented EDITOR based on an earlier "C" version that appeared in the Jan '82 Dr. Dobbs. Since it includes the ability to use some of the features of "smart" terminals, I found its screen handling of scrolls, inserts and deletes to be the fastest I've ever used. A really nice editor that became comfortable to use very quickly and has the minimum essential text editing features -- in fact, I'm using it to prepare this review. Curiously, there are no copy or move commands.

By the way, all these programs came with source, including the PASC compiler. A dedicated customizer could have a ball with these.

Art Weller

Editor's Note: PASC is available from S.E. MEDIA. See advertising this issue.

- - -

TO THE EDITORS OF 68' MICRO JOURNAL

DEAR SIRS,

HEREWITH I SEND YOU 2 NEW FLEX-UTILITIES.
MAYBE THEY COULD BE PUBLISHED IN YOUR MAGAZINE.

1) "RPTERR"
THIS AN OVERLAY TO IMPROVE THE "RPTERR"-ROUTINE OF FLEX. NORMALLY FLEX DOESN'T PRINT THE FILENAME WHEN "RPTERR" IS CALLED; BUT THIS ROUTINE DOES!

2) "PRIORITY"
WHEN YOU DEFINE: ASN S=A, FLEX SEARCHES THE DRIVES ALWAYS IN THE SAME ORDER: 0,1,2,3.
WITH THIS OVERLAY, WHICH IS ALSO A COMMAND, YOU CAN CHOOSE YOURSELF IN WHAT ORDER THE DRIVES MUST BE SEARCHED. FOR INSTANCE, IF YOU HAVE A MEMORY-DRIVE AT DRIVE#2, THEN SAY: PRIORITY,2,0 COMMANDS WILL BE SEARCHED FIRST ON MEMORY-DRIVE, AND IF NOT THERE ON DRIVE#0.
YOU CAN GIVE AS MANY DRIVE#'S YOU WANT, WITH A MAXIMUM OF 4.

BOTH PROGRAMS WERE WRITTEN FOR FLEX09 V3.1

ETIENNE FRANCOIS
DR.M.L.KINGSTR.61
1121 CR LANDSMEER
HOLLAND

RPTERR.TXT

```
*
FREE EQU $7000 JUST AN EXAMPLE
*
* PATCHES FOR IMPROVEMENT OF "REPORT ERROR"
*
RSETIO EQU $CD2A
ERRTYP EQU $CC20
PSTRNG EQU $CD1E
PUTCHR EQU $CD18
*
*
* PATCH "REPORT ERROR" ROUTINE TO THE NEW ROUTINE....
 ORG $D283
 JSR PRERRO
*
*
* PATCH "NOT FOUND" ROUTINE TO GO ALSO TO "REPORT ERROR"
 ORG $D207
```

```
 NOP
 NOP
 NOP
*
 ORG FREE  SOME FREE MEMORY (DRIVERS,SPOOLER ETC...)
*
SAVFCB RMB 2  SAVE CURRENT FCB
*
PRERRO STA ERRTYP
 STX SAVFCB
 JSR RSETIO
 LDX #TXER
 JSR PSTRNG
* PRINT FILE NAME FROM FCB AREA (FCB+4)
* BE CAREFUL, IT MAYBE DAMAGED, BUT SOMETIMES
* THERE'S COPY AT FCB+36; CHECK THAT ONE TOO!
* IF BOTH FILESNAMES ARE DAMAGED PRINT ALL ?'S.
 LDX SAVFCB
 LEAX 4,X  POINT TO FILENAME 1
 LDB #2 CHECK TWO FILENAMES
* CHECK FILENAME 1 OR 2
CHNAME LDA 0,X  CHECK FIRST CHAR OF FILENAME
 CMPA #'A
 BCS DAMAGE  <="A" IS NOT OKE
 CMPA #'Z
 BLS PRTER1  <="Z" IS OKE
DAMAGE DECB
 BEQ NONAME
 LEAX 32,X POINT TO FILENAME 2 (FCB+36)
 BRA CHNAME
* BOTH FILENAMES DAMAGED; POINT TO ALL ????
NONAME LDX #DRQUES ALL ????
 BRA PRTER2
* IF FILENAME IS OKE...
PRTER1 LEAX -1,X POINT TO DRIVE#
 LDA 0,X  GET DRIVE# FROM FCB
 CMPA #3  CHECK DRIVE# IS REAL
 BHI NONAME  IF DRIVE# NOT >0 OR <=3
PRTER2 LDA 0,X+

 ADDA #$30 MAKE DRIVE# ASCII
 JSR PUTCHR  PRINT DRIVE#
 LDA #'.
 JSR PUTCHR
 LDB #8  PRINT FILENAME
 JSR PRT
 LDA #'.
 JSR PUTCHR
 LDB #3   PRINT EXTENSION
 JSR PRT
 LDX SAVFCB
 LDA ERRTYP
 RTS  CONTINUE FLEX
*
TXER FCB #7
 FCC "ERROR WITH: "
 FCB #4
DRQUES FCB '?-$30
TXQUES FCC "????????" FILENAME
 FCC "???"       EXTENSION
*
PRT LDA 0,X+ GET CHAR FROM FILENAME
 BEQ PRT1 DONT PRINT $00
PRT2 JSR PUTCHR
PRT1 DECB
 BNE PRT
 RTS
*
*
 END
```

PRIORITY.TXT

```
* PRIORITY.CMD
* SET PRIORITY OF DRIVES IF ASN S=ALL
*
* FORMAT 1:   PRIORITY  (shows current priority)
* FORMAT 2:   PRIORITY.CMD,[drv1],[drv2],....(max.4)
* FORMAT 3:   PRIORITY.CMD,[drv1][drv2]....(max.4)
*                        (comma's omitted)
*
*
* -----------------------------------------------
*
* PROGRAMMED BY ETIENNE FRANCOIS
*              LANDSMEER, HOLLAND
*
*
*
PSTRNG EQU $CD1E
 TCHR EQU $CD15
WARMS EQU $CD03
SYSDRV EQU $CC0B
DRVSEL EQU $DE0C
PUTCHR EQU $CD18
NXTCH EQU $CD27
LAST EQU $CC11
TTYEOL EQU $CC02
DRVRDY EQU $DE0C DRIVE READY FLEX
*
*
 ORG $C100
*
PRIORT BRA PRIOTT
*
VN FCB 9    23/03/85    E.FRANCOIS
*
COUNT RMB 1
TABPOI RMB 2
*
* INIT DRIVE COUNTER & PRIORITY-TABLE POINTER
PRIOTT LDA #$31
 STA COUNT
 LDX #PRITAB
 STX TABPOI
*
* LOOK FIRST IF DRIVE#'S WERE GIVEN IN COMMAND-LINE
* GET THESE DRIVE#'S FROM COMMANDLINE AND PUT
* THEM IN PRIORITY-TABLE.
*
*
PARAM LDA LAST
 CMPA #$00
 BEQ PRINT NO DRIVES SPECIFIED; PRINT CURRENT PRIORITY.
 CMPA TTYEOL
 BEQ PRINT   SAME....
*
* INIT PRIORITY-TABLE  (4 ENTRIES + END-OF-TABLE)
*
INIT LDX #PRITAB
 LDB #5
INI1 LDA #$FF

 STA 0,X+
 DECB
 BNE INI1 NEXT TABLE ENTRY
*
* GET THE DRIVE#'S FROM CMDLINE
PAR1 JSR NXTCH
 CMPA #$D
 BEQ STOP
 CMPA TTYEOL
 BEQ STOP
 CMPA #$20 SPACE?
 BEQ PAR1 SKIP IT
 CMPA #',   COMMA?
 BEQ PAR1 SKIP IT ALSO
* SO ASSUME IT'S A NUMBER NOW....
 SUBA #$30 CONVERT DRIVE# FROM ASCII TO HEX
 LDX TABPOI
 STA 0,X+ PUT DRIVE# IN PRIORITY-TABLE
 STX TABPOI UPDATE TABLE POINTER
SKIP LDA COUNT
 CMPA #$34
 BCS PAR5 IF <
DUMMY JSR NXTCH   READY; 4 DRIVE'S SPECIFIED;
*                 READ UNTIL $D OR TTYEOL
 CMPA #$D
 BEQ STOP
 CMPA TTYEOL
 BEQ STOP
 BRA DUMMY
*
PAR5 INC COUNT
 BRA PAR1 GET NEXT CHAR; IF THERE'S ANY...
*
* SET SYSDRIVE TO "SEARCH ALL" (S=A)
STOP STA LAST
 LDA SYSDRV
 LDA #$FF
 STA SYSDRV
*
EXIT JMP WARMS
*
*
PRINT LDX #PRITAB CHECK IF AT LEAST 1 PRIORITY
*                WAS GIVEN
 LDA 0,X
 CMPA #$03
 BLS PRINT2 PRIORITY IS DEFINED; PRINT IT
 JMP NOPRIO     ,,     NOT DEFINED; PRINT WARNING
PRINT2 LDX #TXPRI
 JSR PSTRNG
 LDA #5  MAX. 4 DRIVES
 STA COUNT
PRINT1 LDX TABPOI
 LDA 0,X+
 STX TABPOI
 DEC COUNT
 BEQ EXIT MAXIMUM 4 DRIVES!!
```

```
 CMPA #$FF
 BEQ EXIT
 ADDA #$30 WAS HEX; MAKE ASCII
 JSR PUTCHR
 LDA #$20 SPACE
 JSR PUTCHR
 BRA PRINT1
TXPRI FCC "CURRENT PRIORITY: "
 FCB 4
*
NOPRIO LDX #TXNOPR
 JSR PSTRNG
 JMP WARMS
TXNOPR FCB $D,$A,$A
 FCC "PRIORITY WAS NOT YET DEFINED!!!!"
 FCB $D,$A
 FCC "DEFINE IT LIKE THIS:    PRIORITY,0,1"
 FCB $D,$A,4
*
*
* PATCHES FOR PRIORITY ROUTINE; REPLACE OLD FNDNXT
*
 ORG $DDBD OLD PLACE OF FIND NEXT READY DRIVE ROUTINE
*
*
FNDNXT LDX $D40B   - FCB ADDRESS
 LDA 3,X
 LDX #PRITAB
 TSTA  LOOK IF FIRST TIME
 BMI YES   DRIVE# = $FF
NO TST 0,X
 BMI NFOUND END OF TABLE
 CMPA 0,X
 BEQ FOUND
NEXT LEAX 1,X
 BRA NO
FOUND LEAX 1,X
YES TST 0,X
 BMI NFOUND
OKE LDA 0,X
 LDX $D40B
 STA 3,X PLACE IN FCB
 JSR DRVRDY
 BCS FNDNXT DRIVE NOT  READY; NEXT DRIVE
 RTS
*
NFOUND LDX $D40B
 LDB #$10 FORCE ERROR
 SEC
 RTS
*
* PRIORITY TABLE
*
PRITAB RMB 5
*

 END PRIORT
```

PT-69 OS-9

Last year I reviewed several SBCs. One was the PERIPHERAL TECHNOLOGY PT-69 SBC. As you might remember I felt that this was one of the better systems. Well, after a year or so of heavy use, I still feel the same way about this system.

This was one of the SBCs that I installed in a Heath H19 CRT terminal. This system has two 40 track drives. Eighty track drives could be used. It has performed flawlessly and not one bit has ever been lost running FLEX/STAR-DOS. However, as more of our applications are being developed under OS-9, I decided to install the PT-69 OS-9 system.

The PT-69 is designed so as to only run FLEX/STAR-DOS or OS-9. A monitor EPROM is furnished for each system. Conversion from one to the other is simply changing the EPROMs, maybe one or two other simple circuit changes, and away you go. Fact is, it is so simple to change that the complete installation instructions are only one page.

After installing the EPROM a jumper must be changed from the 2716 position to the 2732 position. And if your board is an older one, PT-69 or PT-69-1 you will have to install one wire if you intend to run a Centronics type printer. Also some of the very earliest boards will require one resistor change. And that is it.

Standard level one logical devices are supported /D0, /D1, /TERM, /T1, /P, and /P1.

Available as an option is a CoCo disk driver and device descriptors. Also an upgrade package is very reasonable ($30.00) for those wishing to upgrade older systems to the latest, including the CoCo disk routines and new EPROM. For an additional $15.00 the factory will upgrade your board also.

One very nice feature of this board is a battery backed up time of day system clock. I hate to type in the date and time everytime I fire up. Like the big systems, this one does it for you.

For the person wanting to get into OS-9 and wanting more than a color computer but not the price of some of the other systems, this one is a winner. It is a full blown OS-9 level one system. It easily supports two users in multi-user mode, or allows several applications run in background and interactive at the same time. The cost of our system breaks down as follows:

Used Heath CRT Terminal	$250.00
PT-69 SBC	279.00
OS-9	200.00
2 40 track drives (used)	150.00
Total Cost	$879.00

Mind you, this is all of it, including a very nice 80 character by 25 line CRT terminal. For those already having a CRT, then I would suggest one of their system packages, either floppy or hard disk. The price is hard to beat, and the quality is there. Also we have had remarks from readers commenting on the cooperation received from their phone calls, letters, etc. It is a nice feeling to know that some vendors really do care.

Please see their advertisement this issue.

\- - -

B-A Bergvall, Huskvarna, Sweden, 86-04-06.

A BOOLEAN EPROM PROGRAMMER.
===========================
or
The poor man's PLA programmer.

In a hardware project I needed a general boolean random logic section. The natural choice would have been a PAL programmeable logic device. But, I have neither a PAL programmer nor PAL programming software.

However, an EPROM can perform the function of an unregistred PAL. In fact, the PAL is only a subset of the EPROM. In the PAL only a limited number of product terms can be programmed, but in the EPROM all input combinations are available for programming, each corresponding to one input address.

One disadvantage of the NMOS EPROM is its slow speed, typically 200 to 450 ns, compared to the bipolar PAL 20 to 30 ns. Fortunately, the speed in my application is not very critical, and a 450 ns 2716 NMOS EPROM is speedy enough.

But, how do I program it to solve my boolean equations? I have an EPROM programmer but have never heard of any boolean EPROM software. So I created it, using BasicO9 and my Primal PD2000 O69 computer.

First I named the EPROM inputs and outputs to application specific names. The program "boolrom" was then edited and run to make the EPROM pattern file "eprom1". This file was

copied to the EPROM using my EPROM programmer. The EPROM was inserted into the circuit, completing the EPROM design cycle.

The description below is tailored to my application, but I have tried to make the documentation very explanatory. You can use my program "boolrom" as a template to create your own programs for your combinatorian needs. Just replace my variable names with yours and rewrite the Conditional section with your algorithms.

My 2716 EPROM is used in an application with a step motor and two solenoids and is configured according to Drawing 1. The 2716 has 11 address inputs and 8 data outputs, all used in the application.

The six LSB inputs are connected to a counter, and these inputs form one common integer variable named "count". The remaining five inputs and the eight outputs are used independently and each have a separate boolean variable with its own name.

The allowed mixing of variable types simplifies the algorithm design considerably and illustrates the flexibility of this EPROM programming method. You can think of the added labor involved in specifying the input conditions of the separate counter inputs for the many different counts that are used in the application. When programming PALs you normally have to do that exercise.

Most of the EPROM outputs are fed to flipflops. This causes a problem because the outputs can carry glitches from input transitions causing the flipflops to misbehave. You might considering using another EPROM input as a gating clock. But the EPROM's individual input behaviour is not revealed on the manufacturers' data sheets. An EPROM clock input can consequently not safely gate off other inputs.

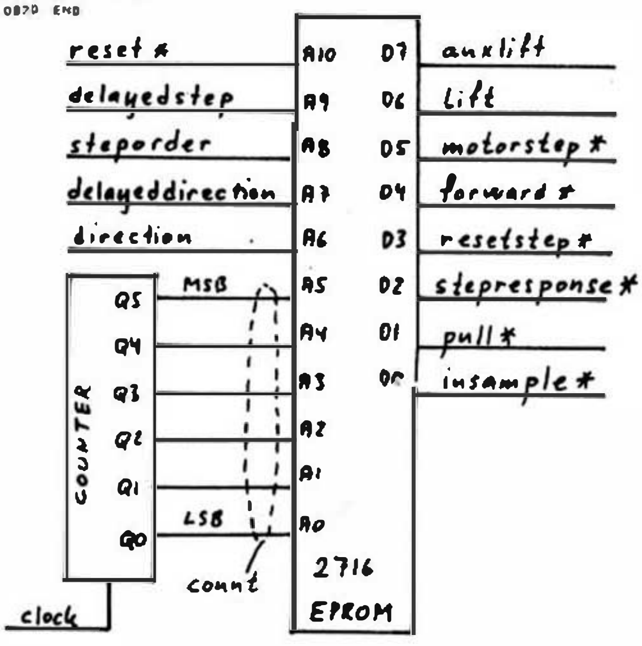
Thus don't use internal clock gating but instead use external clock gating at the outputs to protect flipflops. This is a drawback compared to PALs which produce no glitches when a gating clock is off.

The "boolrom" program is mostly selfexplanatory. It analyses one input address at a time. Each address corresponds to one input combination. For each address the program finds the corresponding values of the input variables. The Conditional section uses IFs, ANDs and ORs on the input variables to create the output variables. The output variables are then collected to an output data byte, which constitutes the data content of that particular input address. The data byte is written to the pattern file "eproml".

The above process is repeated for each input address until the complete address field of the EPROM is covered. The process will take some minutes. The process is slowed down by the repetitive use of power-of-2 expressions. These are used to make the program easier to understand but can be replaced by implicit integers to speed up the process.

```
PROCEDURE boolrom
        REM ***********************************************************
        REM * Program to produce boolean EPROM.                       *
        REM * Bengt-Allan Bergvall, Huskvarna, Sweden 86-04-06        *
        REM * to be freely published and used                         *
        REM ***********************************************************

        REM ****** VARIABLE DEFINITIONS: ******************************
        REM Input variables:
        DIM inputbits,address,count:INTEGER
        DIM A(15):BOOLEAN
        DIM direction,delayeddirection,steporder,delayedstep,reset:
         BOOLEAN

        REM Output variables:
        DIM databyte:BYTE
        DIM D(8):BOOLEAN
        DIM inexample,pull,stepresponse,resetstep,forward,motorstep:
         BOOLEAN
        DIM lift,auxlift:BOOLEAN

        REM Auxiliary variables:
        DIM i,j,k,l:INTEGER

        REM File variables:
        DIM file:BYTE
        DIM filename:STRING

        REM ****** OPEN AN EPROM PATTERN FILE: ************************
        filename="eproml"

        REM Overwrite possible old pattern file:
        ON ERROR GOTO 100
        OPEN #file,filename:WRITE
        ON ERROR
        GOTO 110
100     ON ERROR
        CREATE #file,filename:WRITE
110     REM "Landing point" if old file exists.

        REM ****** CREATE THE EPROM PATTERN: **************************

        REM number of input bits:
        inputbits=11
        BASE 0

 REM ******** Analyze at a time one input address =
 REM *        = one input combination =
 REM *        = one output byte.
 FOR address=0 TO 2^inputbits-1

   REM i=leftover of current input address
   REM j="worth" of particular input address bit
   REM k=bit number of particular input bit
   i=address

   REM Find the input variables A(inputbits-1) to A(0) at the address:
   FOR k=inputbits-1 TO 0 STEP -1

     REM Special! Catch the counter count at A(5)
     IF k=5 THEN
       count=i
     ENDIF

     j=2^k
     IF i>=j THEN
       A(k)=TRUE
       i=i-j
     ELSE
       A(k)=FALSE
     ENDIF
   NEXT k

   REM Assign the input variables:
   reset=NOT(A(10))
   delayedstep=A(9)
   steporder=A(8)
   delayeddirection=A(7)
   direction=A(6)

   REM A(5)-A(0) carrying 6-bit counter value="count".
   REM Inputs available but not used as boolean.
   REM A(5)=MSB, A(0)=LSB.

   REM ******* The Conditional (boolean) section determining
   REM the output variables from the input variables.

   REM Default outputs, used when no specific value is assigned:
   pull=FALSE
   stepresponse=FALSE
   resetstep=FALSE
   forward=TRUE
   motorstep=FALSE
   lift=FALSE
   auxlift=FALSE

   REM In all cases:
   inexample=count=7

   IF reset THEN
     resetstep=count=60
     lift=count<=16 OR count>=39
     auxlift=count>=60 OR count=0
   ENDIF

   IF steporder AND NOT(reset) THEN
     IF direction THEN
       REM forward:
       pull=count>=30 AND count<=61
     ELSE
       REM reverse:
       pull=count>=16 AND count<=39
     ENDIF
     resetstep=count=60
     stepresponse=count=61
     REM stepmotor stepping
     forward=direction
```

```
07CA      IF delayedstep THEN
07D3        REM constant speed
07E4        motorstep=count=3 OR count=7 OR count=14 OR count=18
0804        motorstep=motorstep OR count=24 OR count=28 OR count=
             35
0821        motorstep=motorstep OR count=39 OR count=46 OR count=
             50
083E        motorstep=motorstep OR count=56 OR count=60
0854      ELSE
0858        REM acceleration
0867        motorstep=count=16 OR count=27 OR count=38 OR count=46
0887        motorstep=motorstep OR count=50 OR count=56 OR count=
             60
08A4      ENDIF
08A6    ENDIF
08A8
08A9
08BC    IF delayedstep AND NOT(steporder) AND NOT(reset) THEN
08D3      REM reload the stepmotor
08DB      forward=delayeddirection
08FB      motorstep=count=3 OR count=7 OR count=14 OR count=33
090A      motorstep=motorstep OR count=39
090C    ENDIF
090D
091C    IF NOT(steporder) AND NOT(reset) THEN
0932      REM no steporder, lift:
0918      lift=TRUE
094F      IF delayedstep AND count>=3 AND count<=9 THEN
0955        auxlift=TRUE
0957      ENDIF
0959    ENDIF
095A
099A    REM ******** Transform the output condition to an eight bit byte:
099B
09B5    REM Assign the output bits:
09C1    D(0)=NOT(insample)
09CD    D(1)=NOT(pull)
09D9    D(2)=NOT(stepresponse)
09E5    D(3)=NOT(resetstep)
09F1    D(4)=NOT(forward)
09FD    D(5)=NOT(motorstep)
0A08    D(6)=lift
0A13    D(7)=auxlift
0A14
0A3A    REM Initially, set all output bits low:
0A42    databyte=$00
0A43
0A73    REM Selectively, set the wanted output bits high:
0A8E    REM I= particular output bit
0A9E    FOR I=0 TO 7
0AAC      IF D(I)=TRUE THEN
0ABE        databyte=LOR(databyte,2^I)
0AC0      ENDIF
0AC8    NEXT I
0ACC
0AFB    REM Write the byte to the EPROM pattern file:
0B02    PUT #file,databyte
0B03
0B2F    REM Continue with the next input combination:
0B3A  NEXT address
0B3B
0B75  REM ******** MAKE A NICE ENDING: ***************************
0B7B  CLOSE #file
0B7D  END
```



Drawing 1.

Computer Publishing Center
'68' Micro Journal
5900 Cassandra Smith Road
Hixon, Tennessee 37343

The Public Domain FLEX utilities I've been sending out, called the GIVEAWAY PACKAGE for the lack of a better name, has been updated. Several improvements have been made to CAT, COPY and DISKEDIT, often at the suggestion of Bob Jones. Several additions have been made, including a complete set of utilities for multivolume winchester drives using the WD-1000 or WD-1002-05 winchester controllers. The following catalog shows the dates of the current versions:

SQUEEZE	.TXT	9-21-81	SAVE	.TXT	2-03-82
WORDSWAP	.TXT	3-06-82	DATE	.TXT	4-04-82
LOWERCAS	.TXT	5-23-82	DIRALL	.TXT	5-23-82
CLEANUP	.TXT	5-23-82	PICTURE	.TXT	6-05-82
ALOAD	.TXT	8-28-82	RESMORMG	.LIB	1-01-83
RESMORMP	.LIB	1-02-83	RESIDENT	.TXT	1-02-83
RESIDENT	.DOC	1-02-83	RESPROMP	.LIB	1-02-83
RESLBUGG	.LIB	1-03-83	RESLBUGP	.LIB	1-03-83
RAMLBUG	.LIB	5-04-83	FIND	.TXT	10-10-83
FASTEXEC	.TXT	2-07-84	COPY	.DOC	2-09-84
PDEL	.DOC	11-17-84	RECOVER	.DOC	11-17-84
GEM	.TXT	4-13-85	INITJOE	.LIB	6-14-85
INITGAP	.LIB	6-18-85	INITINFO	.LIB	6-23-85
INITDRV	.LIB	6-23-85	INIT	.TXT	7-13-85
INITABL	.LIB	7-19-85	RAMDISK	.TXT	8-24-85
TEST	.TXT	9-07-85	INITBOOT	.LIB	9-17-85
DISKEDIT	.TXT	9-17-85	DRIVERS	.TXT	10-16-85
WINPARK	.TXT	10-23-85	WINASN	.TXT	10-24-85
DSKALIGN	.TXT	10-29-85	DIRSIZE	.TXT	11-16-85
WINTABLE	.LIB	12-01-85	WINEXAM	.DOC	12-04-85
WINDRV	.TXT	12-08-85	WINFMT	.TXT	12-08-85
WINBOOT	.TXT	12-23-85	GEM	.DOC	12-28-85
WIN	.DOC	12-28-85	CAT	.TXT	12-31-85
PROMPROG	.TXT	1-04-86	CAT	.DOC	1-11-86
PROMPROG	.DOC	1-11-86	COPY	.TXT	1-18-86
HEXCALC	.TXT	2-03-86	DISKEDI2	.TXT	3-16-86

Those who need most of the files updated, or have never sent for the GIVEAWAY package, can send me 6 five inch disks or 2 eight inch disks. Please include a stamped self addressed return container. You need not format the disks, I always record the disks SSSD 35/77 tracks. Those who only need a few files updated can send me a list plus the number of formatted disks needed.

Alternately, the programs can be downloaded from Micheal Evenson's 680X Bulletin Board in California (213) 539-7619. You will need to make two calls a few days apart, the first call to establish your password which allows you access to the XMODEM (CP/M) download section.

Leo Taylor
109 Twin Brook Road
Hamden, Conn. 06514 USA

PERIPHERAL TECHNOLOGY
1480 Terrell Mill Rd., Suite 870
Marietta, Georgia 30067
Telex #880584

404/984-0742

PRODUCT ANNOUNCEMENT

COCO DRIVERS

PERIPHERAL TECHNOLOGY ANNOUNCES AVAILABILITY OF COCO COMPATIBLE DRIVERS FOR OS/9. THESE DRIVERS MAY BE LOADED BY THE USER AND ALLOW READING AND WRITING TO COCO FORMATTED OS/9 DISKETTES. EACH DRIVE TAKES ON A DUAL PERSONALITY AFTER THE DRIVERS ARE LOADED ALLOWING THE DRIVES TO RESPOND AS EITHER /D0 OR /C0, /D1 OR /C1 ETC. THIS ALLOWS EASY DISKETTE INTERCHANGE BETWEEN USERS HAVING A COCO AND A PT69 COMPUTER.

USE OF THE DRIVERS ON PT69 COMPUTERS THAT DO NOT HAVE THE "MICROWARE STANDARD DISK FORMAT OPTION" WILL REQUIRE MODIFICATION OF THE PT69 COMPUTER AND A NEW BOOT EPROM AND OS/9 SYSTEM DISKETTE. THESE DRIVERS ARE NOW INCLUDED WITH ALL NEW ORDERS FOR OS/9 OR SYSTEMS WITH OS/9. AN UPGRADE PACKAGE IS AVAILABLE FOR USERS WISHING TO UPGRADE THEIR PT69 COMPUTER FOR COCO COMPATIBILITY. THE COST OF THIS PACKAGE IS $30. MODIFICATION OF THE USER'S PT69 IS AVAILABLE FOR AN ADDITIONAL $15 IF THE USER DOES NOT WISH TO MODIFY THE BOARD HIMSELF.

FOR ADDITIONAL INFORMATION, PLEASE CONTACT:

PERIPHERAL TECHNOLOGY
1480 TERRELL MILL RD. SUITE 870
MARIETTA, GA 30067
404/984-0742

NOTE: AFTER MODIFYING A PT69 COMPUTER FOR THE NEW FORMAT YOU WILL ONLY BE ABLE TO READ YOUR OLD DISKETTES THAT WERE FORMATTED AS SINGLE SIDED/SINGLE DENSITY DISKETTES.

184 Mont Albert Rd.
Canterbury 3126
Australia.
March 17th 1986

Mr. Don Williams
68 Micro Journal,
5900 Cassandra Smith Rd.
Hixson, TN 37343
U.S.A.

Dear Don,

I have just upgraded my home brewed system to run a 68B09 processor at 2.5 Mhz. I slow the processor down for all addresses above $E000, by using a MMV to stretch phase two of the clock. This stretching of the clock allowed the use of the existing, slower, monitor and I/O devices.

The system has two five inch and two eight inch disk drives and the increase in speed allows double density disk operation for both size drives using program I/O (not DMA).

The Flex disk drivers and the disk boot had to be altered, as so did the disk formatter program.

Whilst working on this upgrade I became very conscious of the number of disk sectors and the interleave factor used by Flex and of course I made good use of the Disk Sector Interleave article by Leo Taylor in the December 1985 issue of Micro Journal.

I still have to settle on the number of disk sectors and the interleave factor that I will finally use.

The attached program is a Basic program that I wrote to compute the disk sector interleaving and eliminate the tedious, pencil and paper task of working out of the numbers.

I have often used programs from Micro Journal and would like to contribute this program for publication, as I think it may be of use to others.

Many thanks for a good 68XX magazine.

Yours Faithfully,

Howard Wills

Howard Wills.

DISK SECTOR INTERLEAVE BASIC PROGRAM

```
 10 REM DISK SECTOR INTERLEAVE
 20 REM
 30 REM HOWARD WILLS      16/03/86
 40 REM
 50 REM VERSION 1.0       16/03,86
 60 REM
 70 REM THIS PROGRAM CALCULATES THE DISK SECTOR NUMBER
 80 REM SEQUENCE FOR A GIVEN INTERLEAVE FACTOR & REPORTS
        ERRORS
90 REM
100 DIM S(99)
110 REM PRINT HEADER INFORMATION
120 GOSUB 640
130 REM PRINT SECTOR INFORMATION
140 GOSUB 710
150 REM GET REQUIRED SECTOR NUMBER
160 GOSUB 820
170 IF N>99 THEN 160
180 REM GET REQUIRED INTERLEAVE FACTOR
190 GOSUB 880
200 IF I>=N THEN 190
210 FOR A=1 TO N
220 S(A)=0
230 NEXT A
240 A=1 : H=0 : SN=2
250 E=0
260 S(A)=1
270 REM NOW STEP TO INTERLEAVE
280 A=A+I
290 IF A<N+1 THEN 310
300 A=A-N
310 IF E<>0 THEN 610
320 IF S(A)<>0 THEN 570
330 S(A)=SN
340 SN=SN+1
350 IF SN<>N+1 THEN 280
360 REM PRINT SECTOR PATTERN
370 FOR Z=1 TO N STEP 18
380 X=Z
390 Y=Z+17
400 IF Y>N THEN Y=N
410 GOSUB 460
420 PRINT
430 NEXT Z
440 GOSUB 940
450 GOTO 150
460 PRINT
470 FOR A=X TO Y
480 IF A<10 THEN PRINT" ";A;
490 IF A>9 THEN PRINT A;
500 NEXT A
510 PRINT
520 FOR A=X TO Y
530 IF S(A)<10 THEN PRINT" ";S(A);
540 IF S(A)>9 THEN PRINT S(A);
550 NEXT A
560 RETURN
570 H=H+1
580 IF H=N THEN E=1
590 A=A+1
600 GOTO 310
610 PRINT
620 PRINT"*** BAD INTERLEAVE FACTOR - PROGRAM ABORTED ****"
630 GOTO 150
640 PRINT
650 PRINT TAB(5);"DISK SECTOR INTERLEAVE PROGRAM BY HOWARD WILLS"
660 PRINT TAB(5);"**********************************************"
670 PRINT
680 PRINT"MAXIMUM NUMBER OF SECTORS ALLOWED = 99"
690 PRINT
700 RETURN
710 PRINT"NORMAL FLEX DISK SECTOR VALUES ARE   "
720 PRINT
730 PRINT"5 INCH S.D. = 10"
740 PRINT
750 PRINT"5 INCH D.D. = 18"
760 PRINT
770 PRINT"8 INCH S.D. = 15"
780 PRINT
790 PRINT"8 INCH D.D. = 26"
800 PRINT
810 RETURN
820 PRINT
830 PRINT"ENTER REQUIRED SECTOR VALUE = ";
840 INPUT N
850 IF N<>0 THEN RETURN
860 PRINT"**** CAN'T HAVE ZERO SECTORS ****"
870 GOTO 820
880 PRINT
890 PRINT"ENTER REQUIRED INTERLEAVE NUMBER = ";
900 INPUT I
910 IF I<>0 THEN RETURN
920 PRINT"**** CAN'T HAVE ZERO INTERLEAVE ****"
930 GOTO 880
940 PRINT
950 PRINT"THERE WERE ";H;" SECTOR CLASHES"
960 PRINT
970 RETURN
```

WESTCHESTER Applied Business Systems, Inc
2 Pea Pond Lane, Briarcliff Manor, New York ?510

Dear Don, Larry and Tom,

Several Users have inquired as to the differences between XDMS levels I, II and III - and the new XDMS-IV system. I am sure you have received similiar questions. Let me attempt to briefly highlight the differences and some of the advantages of the new system.

Before doing this, I'll restate the purpose of these systems! They are designed to allow users to structure, enter, retrieve, process and format data. The user views file data as a tabular listing, with column (field) headings and rows of records. Any field is a potential sort key, selection key, or match key to other files. Data fields may be selected, reordered, merged from other files, and reformatted on output. Records may be sorted, organized into subsets, and inserted into report forms. Up to three files may linked as a relational database on report output. Many other features exist, but for now let's concentrate on the new XDMS-IV system.

First of all, XDMS-IV is a single, integrated program. Once invoked, all commands are in memory and instantly available. With old XDMS, the User would first invoke the nucleus, and then enter each command from the FLEX +++ prompt. There is a few second delay while each command loads from disk. With XDMS-IV, the User enters commands from a "X!" prompt and there is no delay.

XDMS-IV is designed to be "session oriented", that is! The User does most of his/her work without exiting to FLEX. For this reason, We've built in DIR, LIST, COPY, RENAME and DELETE utility commands and a text editor. The database CREATE and UPDATE utilities are also integral and execute instantly. The REPORT and GENERATE functions disappear with XDMS-IV and become integral to the command interpreter. For ad hoc reports the user simply enters the necessary processing statements and a "RUN" command.

XDMS .CTL files are replaced by enhanced .PRC process files. Processes may execute utility commands, process commands, other processes or themselves. Menus and prompts may be output to the screen, and commands or processes may be conditionally executed based on operator selection.

The heading, form, body and footing text blocks which surround the insert, link and main file records on reports are replaced by a single integrated form command. Record or field placement is a function of indicators placed in the form text. Forms may be coded in-line, or may reside in a separate file. Form files may be used by multiple processes.

Coding of output page size, margins, etc. for each process is eliminated by user preset attributes for the terminal and printer. These are automatically invoked depending on the output routing, and may be overridden if required. XDMS-IV also supports cursor positioning, home and clear, highlighting, etc. as available on most terminals. Printer boldface, underlining, double width and italics are supported. Control codes may be output to select font sizes or to configure other printer or terminal options prior to report output.

XDMS-IV is a completely new architecture with many other features. The highly efficient code occupies slightly more space than the XDMS nucleus and GENERATE together, yet provides all functions. The new manual contains an alphabetically organized command reference section, where each command is explained and illustrated.

I've attached a comparison matrix for other XDMS attributes. Naturally, I used XDMS-IV to create and print the matrix. Specific capabilities are indicated with a "Yes" or "No", and some are noted with a "Mod", meaning "separate module" or a "Int" meaning "integral command". The latter implies that the function may be executed within a process.

Because of the implementation effort involved, We view XDMS-IV as a totally new system, and not an upgrade. It's $350.00 list price is a bargain when one realizes that it is comparible to popular PC software selling for much more. We do offer a migration incentive (discount) to current XDMS, XDMS+, and XDMS level I, II and III users. These users may order XDMS-IV for $250.00 with valid proof of purchase. Orders may be placed with Southeast Media, 5900 Cassandra Smith, Hixson TN 37343.

Best Regards,

Bill Adams

* * * XDMS COMPARISION * * *

ATTRIBUTE------------------------	XDMS-LI	XDMS-L2	XDMS-L3	XDMS-IV
Architectural Modular/Integrated	Mod	Mod	Mod	Int
Number of modules	4	5	16	1
Position Independent Code	No	No	No	Yes
Relocatable	No	No	No	No
Session Oriented	No	No	No	Yes
Maximum Fields/Groups per Record	24	24	24	32
Maximum Label Characters	8	8	8	12
Maximum Record Size	256	256	256	1024
User Install Term/Print Attrib's	No	No	No	Yes
Dynamic Set Output Attributes	Yes	Yes	Yes	Yes
Virtual Paging to Dedicated Disk	Yes	Yes	Yes	Yes
Virtual Paging to Random File	No	No	No	Yes
Terminal Cursor Control Support	No	No	No	Yes
Printer Special Feature Support	No	No	No	Yes
Process Control Files	Yes	Yes	Yes	Yes
Execute a Process from a Process	No	No	No	Yes
Operator Menu/Process Selection	No	No	No	Yes
Create, Update & List File	Yes	Yes	Yes	Yes
Maximum Record Selection Tests	15	15	15	15
Equal, Less, Greater Tests	Yes	Yes	Yes	Yes
Starts-with/Masked Tests	Yes	Yes	Yes	Yes
Contains-string Test	No	No	No	Yes
Not Starts-with/Contains Tests	No	No	No	Yes
Tests against other fields	No	No	No	Yes
Select Specific Fields for O/P	Yes	Yes	Yes	Yes
String Multiple Fields into One	No	No	No	Yes
Redefine field type/len on O/P	No	No	No	Yes
Merge fields from Auxiliary File	Yes	Yes	Yes	Yes
Join Records from Two Files	No	No	No	Yes
Array/classify data into columns	No	Yes	Yes	Yes
Sum records/arrays by key value	No	Yes	Yes	Yes
Un-array columns into records	No	No	No	Yes
Calculations on field data	Yes	Yes	Yes	Yes
Maximum Sort Fields	15	15	15	15
Insert File as Report Heading	No	Yes	Yes	Yes
Link Three Files as M:1:M on O/P	No	Yes	Yes	Yes
Invert File Relationship on O/P	No	No	No	Yes
Insert Records within form text	No	Yes	Yes	Yes
Insert record fields into text	No	Yes	Yes	Yes
Integrated Forms & Form Files	No	No	No	Yes
Output Report to File	No	Yes	Yes	Yes
Output to Database File	No	Yes	Yes	Yes
Output Database to ASCII File	No	No	Mod	Int
Input ASCII file to Database	No	No	Mod	Int
Purge Database File	No	No	Mod	Int
Plot (Bar Graph) Function	No	No	Mod	No
Display File Format	No	No	Mod	Int
General File Utilities	No	No	No	Yes
Text Editor	No	No	No	Yes

NEW BRITISH COMPETITOR TO COMMODORE AMIGA

Micro Concepts launch new Microbox III at Microprocessor Development Show

Micro Concepts launch their new Microbox III - a British made competitor to the Commodore Amiga and the Atari ST520 - at the Microprocessor Development Show, Wembley Conference Centre, February 11-13. Micro Concepts are at 2 Stephens Road, Cheltenham, Glos, GL51 5AA, telephone (0242) 510525.

Microbox III is a low cost, versatile single board colour graphics computer, based on the fast operating 68000 processor. It is available as a complete system with power supply, disk drives, monitor and keyboard, with power supply and disk drives only, or as an assembled and tested board for OEM applications. Prices start at £650 + VAT, putting the Microbox III easily within reach of the personal computer user.

The system is based on a 220 x 230 mm eurocard, and contains 512k bytes of ram, up to 192k bytes of eprom, medium resolution colour graphics and a wide variety of mass storage and communication peripherals. Special features include exceptional colour graphics with gen-locked video, interfaces for twin 800k byte floppy and 20 megabyte hard disks, up to 24k bytes of CMOS battery-backed static ram, a Smart real time clock, parallel and serial ports and a mouse interface - while the choice of operating systems includes TRIPOS, GEM (CPM 68k), OS9-68k and the QDOS-compatible SMS-2. GEM and TRIPOS are compatible, respectively, with the Atari and Commodore systems. Software in the form of systems services and a WIMP type user interface is provided as standard.

Unlike the American systems, it is designed to use the PAL video standard (though the NTSC standard can be used) giving higher screen resolution and proper gen-locking circuitry. Microbox III also has a higher bus band width: 90 - 100%, compared with 5 -

100% for the Amiga, and it has a wider choice of display modes than either machine.

The main memory is provided by 16 256k bit d-rams, giving 512k bytes - shared between the processor (running at 8 MHz for 90% of the time) and the display, using an interleaved scheme which gives the processor access to the ram at any time without disturbing the display. The display uses 80k bytes, leaving 432k bytes available. Two areas of battery-backed CMOS static ram are also provided, holding up to 24k bytes and with a battery life of up to 10 years; one is combined with the eprom sockets, the other with the real time clock. The latter can hold system parameters - character sets etc., and is protected against system crashes.

The four 28 pin eprom sockets can hold up to 192k bytes of eprom. Two sockets may be populated with 27 512 eproms, with 128k bytes of systems software. The remaining two can be configured either for two 27256 eproms or, with the addition of two Smart sockets, two 8k or 32k CMOS static ram. By changing links these two sockets can also be configured to be an on-board 27256 eprom programmer, allowing the board to act as a stand-alone development system.

Display is based on the Motorola Raster Management System chip set, giving either 80 x 25 text with user defineable character sets, or up to 640 x 500 with 4 colours or 320 x 215 with 16 colours. Look up tables give 32 colours out of a range of 4096, on display at any one time. Both vertical and horizontal smooth scrolling is available, and the video can be gen-locked or synchronised to an incoming composite video input, allowing the board's graphic ouput to be mixed or overlaid with the incoming video.
Other features include a dual 5 1/4 " or 3 1/2 " floppy disk drive interface, using the Western Digital 1772 device and giving a total of 800k bytes on a single drive, or 1.6 megabytes for the whole system. There is also a SASI buss interface to connect to a 3 1/2" 20 megabyte Winchester disk drive.

MICRO CONCEPTS
2 St Stephens Road
Cheltenham
Glos GL51 5AA
Tel: (0242) 510626

Classifieds

Winchester 10 Megabyte Drives

Two (2) 10 Megabyte Hard-Disk Winchester Drives. Working - were removed for upgrade to larger drives.

1 - RMS Model #509 $275.00
1 - Seagate Model #412 $275.00

(615) 842-4600 Tom 9-5 EST.

LSI 68008 CPU card, "C" Compiler and Digital Research CPM/68K $350.

Tano Outpost II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $595.

MICROKEY Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertised on p. 51 DEC. 84 68' Micro Journal. $1800.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS. $745.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $359.00 ready to run.

S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1290.

1-CDS1 20 Meg Hard Disk System with Controller $1000.

(615) 842-4600 M-F 9 AM to 5 PM EST

***

68008 HARD DISK SYSTEM - COMPLETE

512K 68008 system, 10 megabyte hard disk, Xebec 1410A HD controller, 80 track double side, double density floppy. Complete with cabinet/power supply. Taken in on Mustang-020 trade-in. Version 1.2 OS-9, Basic09, Stylo, Mail Merge, Spelling Checker, Dynacalc - like new - original price $2,900.00 range (advertised) - **SPECIAL - ONLY $1750.00.**

615 842-4600 - Data Comp, ask for Don or Tom.

***

WANTED blank Data Systems 68 boards. Alen Gordon, M.D. 1435 W. 49th Place, Hialeah FL 33012 (305) 822-1100

***

Stride 420 computer system. Wyse 50 terminal; 2 Megabytes RAM; 2 640 KByte 5 1/4 inch floppy disk drives; 10 Mhz 68000 CPU; VMEbus compatible; 4 serial ports; Centronics printer port; P-System with Pascal and 68000 assembler. $2500 or make offer. Call Steve (213)379-3457

***

Wanted for use with Swtpc 6809: DMF 3 Board, Memory Board, Hard Disc Controller, Hard Disc Unit, Printer Buffer, Details Voltage Hertz and prices, to: Coleman, P O Box 305, Feilding
New Zealand

SPERRY NEWS

****

Introduces Two High-Performance 32-Bit Series 5000 Systems

BLUE BELL, PA — Sperry introduced two new members of the Series 5000 family of microprocessors utilizing the UNIX (TM) operating system - the Model 50 and the Model 90.

Broadening Sperry's micro-to-mainframe product line supporting the UNIX OS, the new systems utilize 32-bit MC68020 (TM) technology and offer improved performance over existing Series 5000 models.

The 5000/50 and 5000/90 offer full 32-bit capability with CPU performance over two times their Series 5000 predecessors. Utilizing an optimized UNIX System V operating system and C compiler for the MC68020, both systems are functionally compatible with other Series 5000 members. Users can move applications, horizontal software tools, programming languages and interconnect software across systems without recompilation.

Both systems provide a common interface to industry standard software, peripherals and communications facilities supported by other Series 5000 products. The systems function as stand-alone or departmental systems supporting SNA, X.25, Ethernet (R) and other industry standard data communications protocols. The industry standard design enables users to maximize hardware investment and remain flexible when reconfiguring or enhancing systems.

In addition to the new models, Sperry announced upgrade capability for existing Series 5000 systems, Models 40, 60 and 80, introduced in November, 1984. The earlier models may be field-upgraded through installation of the 32-bit MC68020 CPU and a revised operating system.

<u>Pricing and Availability</u>

The Series 5000/50 basic hardware ranges in price from $23,030 to $28,530 and is currently available. Basic Series 5000/90 hardware ranges in price from $68,300 to $74,300 and will be available in June, 1986. System options, workstations and software licensing fees are priced separately.

MC68020 field upgrade capability for existing Series 5000/40, 5000/60 and 5000/80 range in price from $6,995 to $12,000 and will be available in July, 1986.

All products are sold through Sperry's worldwide sales organization and authorized Sperry dealers.

<u>Sperry Series 5000, Model 50</u>

The Series 5000/50 can support up to 32 interactive users. The CPU with 8 kilobytes (KB) of data and program cache memory and 2 to 16 megabytes (MB) of main memory (RAM), improves processing performance and allows the execution of large, complex programs. An optional MC68881 Floating Point Processor can be added.

A minimum configuration includes a 68020 processor, 8 KB cache memory, 2 MB main memory, 85 MB fixed disk, 45 MB streaming cartridge tape, 8 async RS-232 ports, 2 synchronous ports, 1 parallel port, sync line interface module and power supply with battery back-up.

The 5000/50 supports up to two expansion cabinets. One Small Computer System Interface (SCSI) tape drive controller and one 1600/3200 bpi tape drive are standard with each cabinet. A 10 MHz disk drive controller and up to two 340 MB fixed disk drives are optional, enabling the 5000/50 to grow to a maximum system capacity of 1.6 gigabytes (GB).

<u>Sperry Series 5000, Model 90</u>

The Series 5000/90 can support up to 88 interactive users and features the ability to install up to four 68020 processors for expansion. Memory ranges from 4 to 16 MB and the system supports 8 KB of data and program cache memory.

A MC68881 Floating Point Processor is standard. This processor permits hardware acceleration of mathematical calculations.

A 5000/90 minimum configuration includes a 68020 processor, Floating Point Processor, 8 KB cache memory, 4 MB main memory, 160 MB fixed disk, 45 MB streaming cartridge tape, 4-channel Direct Memory Access (DMA) controller, 16 sync/async ports, 2 parallel ports and power supply.

The 4-port DMA Controller allows attachment of four disk/tape controllers permitting a maximum of sixteen disks and eight tapes to be configured. Disk types range from 160 MB (processor cabinet only) to 515 MB drives (expansion cabinet only). Up to four expansion cabinets are permitted for a maximum capacity of up to 8 gigabytes (GB).

Sperry's UNIX OS Product Line

Sperry's micro-to-mainframe family of Personal Computers, 5000 Series of multi-user microcomputers, the Sperry 7000/40 super minicomputer and the Sperry 1100 Series offer the industry's largest range of computing power based on AT&T's UNIX System V Operating System. The UNIX OS compatibility allows orderly system growth, protects hardware and software investment and provides unparalleled applications portability — from the desk-top all the way to the computer room.

Sperry is a leading manufacturer and supplier of electronics-based, high-technology systems and services for commercial business and government, defense, aerospace and maritime markets. Sperry's computer operations serve more than 18,000 customers in 50 countries and, in fiscal year 1985, reported revenue of $4.15 billion, part of a total corporate revenue of $5.7 billion.

###

RELEASE NO: 486/3259

Sperry is a trademark of Sperry Corporation
UNIX is a trademark of AT&T Bell Laboratories Inc.
MC68020 is a trademark of Motorola Corporation
ETHERNET is a trademark of Xerox Corporation

M68000
MICRO MINUTES

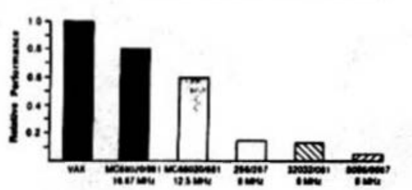
MM- 551-02

REAL-LIFE PERFORMANCE OF THE MC68881

Now that several third party software vendors have had a chance to optimize their Fortran compilers to produce in-line MC68881 object code (versus either subroutine or trap calls), we thought it might be interesting to once again perform some Whetstone comparisons between these compilers and other machines. The following table encapsulates the results, and proves conclusively that the MC68881 delivers the best floating point performance of any microprocessor implementation!

Configuration(Sys)	[redacted]	Speed (MHz)	KWhets
~~VAX 11/780 (w/ FPA[1])~~	VMS/DEC Fortran	---	1,250
SUN-3 (020/881)	4.2bsd/f77 rel 3.1	16.67	1,030
MC68020/881(BM20)[2]	Unisoft Unix™ V/SVS[3]	16.67	1,000
MC68020/881(BM20)	Unisoft Unix™ V/SVS[3]	12.5	780
80286/80287(310)[4]	Xenix/Intel Fortran	8.0	193
NS32032/081[5]		8.0	168
8086/8087(310)	Xenix/Intel Fortran	8.0	63

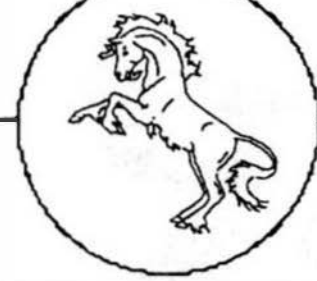
1 "Floating Point Accelerator" (board set ($14K))
2 Motorola "Benchmark 20"
3 Silicon Valley Software Optimized Fortran Compiler
4 Intel 286/310-41ES ("Engineering Special")
5 Hardware/System/Compiler unknown -- from published material by National



MOTOROLA
Austin, Texas

COPROCESSORS

MC68881 PERFORMANCE



MUSTANG-020 & UniFLEX 6809/68020 X-TALK
A C-Modem/Hardware Hookup

Exclusive for UniFLEX MUSTANG-020 is a new package from Data Comp (CPI). X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTP 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020.

The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020.

The X-TALK software is furnished on two disks. One eight inch disk contains the S.E. MEDIA modem program C-MODEM (6809) the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transfered between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also.

X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only

X-TALK Complete (cable, 2 disks)	**$99.95**
X-TALK Software (2 disk only)	**$69.95**
X-TALK with CMODEM Source Included	**$149.95**

Order from:

Data Comp Division (CPI)
5900 Cassandra Smith Rd.
Hixson, TN 37343
615 842-4601
TELEX 510 600-6630

Note: MUSTANG-020 current owners must furnish serial number from back plate of MUSTANG-020 system for this special offer.

MICRONICS
RESEARCH CORP.

Dear Don,

Last time I wrote, I almost got side-tracked into a discussion of the INT(X) function, which in turn would have led me on to Integer Variables. You may wonder what I could possibly say about these that you don't already know, but I'll have a go at it anyway.

To begin with, let's agree that the INT of a non-integer Floating-Point number is the integer **immediately BELOW that number.** That is to say, INT(5.7) = 5 and INT(-5.7) = -6. It is unfortunate that the KBASIC compiler (**not XBASIC**) has chosen to change this definition for negative numbers, thus making their INT(X) into a TRUNCATE(X). To continue with our discussion, the numbers 1, 13, -15, etc are Integer constants, while AX, BX, and CX are Integer Variables in the range +/- 32767, and A, B and C are Floating-Point variables (FP variables, for short).

OK, now let's get down to some serious business. You'll find that lots of programs can be speeded up (quite apart from taking up less room in memory) if you make as many variables as possible into Integers.

PAT

with C Source

$229

*** PAT from Southeast Media -- A full Feature screen oriented **TEXT EDITOR** with all the best of "PIE". For those who swore by and loved only PIE, this is for you! **All PIE features and much more!** Too many features to list. And if you don't like these, change or add your own. C source furnished. Easily configured to your CRT, with special config section.

68008 - 68000 - 68010 - 68020 OS-9 68K **Special: $229.00**
Includes full C source

68008 - 68000
COMBO - PAT JUST $249
68010 - 68020

*** JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the **Display**. "**User Configurable**" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with **PAT** or any other editor. The ONLY stand alone text processor for the 68XXX available, and at a very low price

68008 - 68000 - 68010 - 68020 OS-9 68K **Special: $79.95**
Includes full C source!

$79.95

with C Source

JUST

As a case in point, let's consider the simple loop :

```
10  FOR I = 1 TO 10
20    PRINT I;
30  NEXT I
```

RUNning this program will produce the following :

```
1  2  3  4  5  6  7  8  9  10
```

If the 'I'a are changed to 'I%', the result will be the same, only it will happen faster.

Now let's add another line to the original program, and RUN it again.

```
25     IF I/3 = INT(I/3) THEN PRINT "BOING!";
```

This time we'll see

```
1  2  3 BOING! 4  5  6 BOING! 7  8  9 BOING! 10
```

but not if we change all 'I's to 'I%' as we did for the first example. This time we'll see 'BOING!' printed out after each and every number. What went wrong??

If we look at I/3 and INT(I/3) for the first program, we have

I	1	2	3	4	5	6	7	etc
I/3	.33	.66	1	1.33	1.66	2	2.33	etc
INT(I/3)	0	0	1	1	1	2	2	etc

causing a match at every 'I' which is exactly divisible by 3, whereas I%/3 and INT(I%/3) are both identical to the line for INT(I/3) above. How can we preserve the intent of the original program, and STILL change our 'I's to 'I%'? If we keep in mind that I%/3 produces the sequence 0,0,1,1,1,2 only becamae I% and 3 are both Integers, the solution is to make one of them a Floating-Point Integer, which will produce a FP result. We obviously can't do this with I%, but we can change the '3' to '3.' (the added decimal-point indicating that '3' ie really a FP '3.00') to make Line 25   IF I%/3. = INT(I%/3) THEN PRINT "BOING!"; and we've achieved the desired result. Of course, an astute reader would also observe that we no longer need the INT() part either, so Line 25 ends up as

```
25     IF I%/3. = I%/3 THEN PRINT "BOING!";
```

Another candidate for conversion is the pattern :

```
X = INT(RND(0) * 52 + 1)
```

which selects a random integer in the range 1 - 52, as part of a card-shuffling routine, let's say.

We simply change this to read X% = RND(0) * 52 + 1.

But there are many pitfalls awaiting the unwary - - especially where division is involved (as in our first example) or if we venture into the domain of negative numbers. Also what do we do, if instead of I%/3 (which was easy to change to I%/3.) we have the situation I%/J%, where it is impossible to tack a decimal-point onto either Integer??

Let's examine these different possibilities. Suppose we consider one line of a program :

```
100  A = INT(B / 4 * C)
```

where examination of the complete program shows that A, B, and C are always integers. A prime candidate for conversion to A%, B%, and C% (and elimination of INT) you'd think. But if we assign, let's say, the values B = 7 and C = 5 then Line 100 as given would produce the result A = 8, because INT(7 / 4 * 5) = INT(8.75) = 8. On the other hand A% = B% / 4 * C% would give the result as 5. (Why?) Changing the '4' to '4.' as above produces the correct result of 8.

But what if we were to replace the constant '4' in our Line 100 with the Variable D, and assign the value 4 to D? It makes no difference to Line 100 as it stands, but in our transformed Line we would have

```
A% = B% / D% * C%
```

which would again give the wrong result of 5. What to do? We now have nowhere to tack on a decimal-point!!

A quick and dirty solution is to change the order of the variables on the right-hand side of the equation to give A% = B% * C% / D% (try it, and compare), where division is now the last operation to be carried out. This, however, will only work if the partial product, B% * C%, does not exceed the range +/- 32767, and provided the end result is non-negative. Try setting B = -7, and compare with the case of B% = -7.

A more elegant solution would be to preserve the order of the variables, but to multiply the <u>first</u> variable on the right-hand side by the identity for multiplication. This is equal to '1' (or '1.' in our example) as multiplying (or dividing) a number by '1' leaves it unchanged, i.e. it preserves its identity. Similarly the identity for addition (or subtraction) equals '0', as adding 0 to (or subtracting it from) a number leaves the number unchanged. Now we have

```
A% = B% * 1. / D% * C%
```

which should satisfy the transformation. On the other hand, as addition is faster than multiplication it would make more sense to write it as

```
A% = (B% + 0.) / D% * C%
```

Well, who would have thought there was so much to write about INTEGERS? There's no doubt that I've overlooked a few problem areas somewhere, but this discussion should at least pinpoint some of them and perhaps offer some guidelines for overcoming them. So tread warily when you venture into the field of Integer-Variables and the INT() function - don't make any assumptions as to what the result will be, but test your assignments and expressions for a range of conditions before making them permanent.

At this point, I'm not sure what I'll talk about next time as I seem to have run out of steam on XBASIC. However, we'll see!

Sincerely,

Bob

R. Jones
33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA. V2S 1E2

SK*DOS

(formerly called STAR-DOS) is now available for both **68000 and 6809** computer. The same great DOS, but now better than ever, with enhancements which make it ideal for 6809 users moving to the 68000/68008/68010/68020. Available off-the-shelf now for the Emerald ESB-I and ESB-II computers, (others soon), and for licensing to OEMs at attractive terms. Single copies to end users are $75 (6809 version) and $125 (68K version). Configuration Manual (optional at $50) gives full details on adapting to new systems, supplied FREE to SK*DOS/68K purchasers until September 1. Adapt SK*DOS to a new system and receive a royalty on your adaption! Call us at 914-241-0287 for more information. See STAR-DOS Ad on page 56.

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4600
For Ordering
Telex 5106006630

Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

SOFTWARE DEVELOPERS!

YOU'VE JUST BEEN GIVEN THE BEST REASON YET
TO GET OUR 68000/UNIX® DEVELOPMENT SYSTEM

THE VAR/68K® SERIES

[illegible] **$10.100**
Includes Terminal, 20 Mb hard disk, 512K RAM, 8 ports and REGULUS®

[illegible] **$12.900**
Includes all of above, plus 20 Mb tape streamer

RESELLERS!

IBM PC/OS9 Compatibility

Smoke Signal can add IBM PC capability to your OS9 system for as little as $1195 plus software.

TO OBTAIN YOUR VAR/68K
AT THESE LOW PRICES, CONTACT:

SMOKE SIGNAL
WESTLAKE VILLAGE, CA 91362
(818) 889 9340 / Telex 910-494-4965

Clearbrook Software Group

CSG IMS Information Management System

Some notable features include:

- General purpose database manager.
- Menu-driven front end.
- Comprehensive applications language.
- User definable screen forms.
- Interactive ad-hoc query environment.
- User definable report forms and report generator.
- Data base program generator.

CSG IMS for OS9/6809 LII is $495.
Introductory price until June 30, 1986 is $395

A run-time package for user-developed and distributed applications is $100.

CSG IMS will be available for OS9/68000 and OS9/6809 LI in the second quarter of 1986.

Other CSG Products:

Libr - this is an object librarian, designed to create, inspect and maintain modules and libraries. For use with Microware's C compiler and RMA assembler.
For OS9/6809: $50

Tx - is a general purpose text editor, and has features making it suitable for program creation and editing. It is the same editor which is included with CSG IMS.
For OS9/6809 (soon for OS9/68000): $50

For information or orders, write:

Clearbrook Software Group
446 Harrison Street
PO Box 8000-499
Sumas, WA USA 98295-8000
Telephone: (604)853-9118
Dealer inquiries welcome.

North American orders add $5 for shipping.
Foreign orders add $10 for shipping.

OS9 is a registered trademark of Microware and Motorola

Hard Disk Subsystem for SS-50 Computers

This proven subsystem adds hard disk speed and storage capacity to your computer yet requires only one SS-30 slot. Software (with source) is included for your choice of FLEX9* or STAR-DOS*, OS-9* Level I or Level II, or OS-9 68K operating systems. The software honors all operating system conventions. The software is designed for the Xebex S1410 controller interfacing to any hard disk drive that conforms to the ST506 standard. Four subsystems are available:

1) 27 MB (formatted) WREN* hard disk, Xebec S1410A controller, SS-30 interface card, all cables, and software for $2850;
2) 7.5 MB (formatted) Tandon TM-603 hard disk, rest same as above for $895;
3) no hard disk, rest same as above for $600; and
4) SS-30 interface card and software for $200.

All prices include shipping. We accept Visa and Mastercard without adding a surcharge. Texas residents must add sales tax. The subsystem may be mounted within your computer chassis or in a separate enclosure with power supply. Please write or phone (include your day and evening phone numbers) for more information. We will return North America calls so that any detailed answers will be at our expense.

WELLWRITTEN™ ENTERPRISES

M/C VISA

P.O. Box 9802 - 845
Austin, Texas 78766

Phone
512 335-9446

FLEX is a trademark of Technical Systems Consultants, Inc.
OS-9 is a trademark of Microware and Motorola
WREN is a trademark of Control Data Corporation

SK *DOS

(formerly called STAR-DOS)
is now available for both

68000 and 6809

computers. The same great DOS, but now better than ever, with enhancements which make it ideal for 6809 users moving to the 68000 / 68008 / 68010 / 68020. Available off-the-shelf now for the Emerald ESB-I and ESB-II computers, (others soon), and for licensing to OEMS at attractive terms. Single copies to end users are $75 (6809 version) and $125 (68K version). Configuration Manual (optional at $50) gives full details on adapting to new systems, supplied FREE to SK*DOS/68K purchasers until Sept. 1. Adapt SK*DOS to a new system and receive a royalty on your adaptation! Call us at 914-241-0287 for more information.

STAR-KITS SOFTWARE SYSTEMS CORPORATION Box 209 Mt. Kisco NY 10549

ANDERSON COMPUTER CONSULTANTS & Associates

Ron Anderson, respected author and columnist for 68 MICRO JOURNAL announces the **Anderson Computer Consultants & Associates**, a consulting firm dealing primarily in 68XX(X) software design. Our wide experience in designing 6809 based control systems for machine tools is now available on a consultation basis.

Our experience includes programming machine control functions, signal analysis, multi-axis servo control (CNC) and general software design and development. We have extensive experience in instrumentation and analysis of specialized software. We support all popular languages pertaining to the 6809 and other 68XX(X) processors.

If you are a manufacturer of a control or measuring package that you believe could benefit from efficient software, write or call Ron Anderson. The fact that any calculation you can do with pencil and paper, can be done much better with a microcomputer. We will be happy to review your problem and offer a modern, state-of-the-art microcomputer solution. We can do the entire job or work with your software or hardware engineers.

Anderson Computer Consultants & Associates
3540 Sturbridge Court
Ann Arbor, MI 48105

Heavy Duty
Switching Power Supply

For a limited time we will offer our HEAVY DUTY switching power supply, for those of you wanting to do your own thing with hard disk systems, etc. Note that this is a price far below normal prices for supplies of this quality.

MAKE: Boschert

Size: 10.5 x 5 x 2.5 inches-including heavy mounting bracket/heatsink.

Rating: in 110/220 ac (strap change) Out: 130 watts

Outputs: +5 volts - 10.0 amps
+12 volts - 4.0 amps
+12 volts - 2.0 amps
-12 volts - 0.5 amps

Mating Connector: Terminal strip — Load reaction: Automatic short circuit recovery

Price: $59.95
2 or more $49.95 each

ADD $ 7.50 each for S/H

MAKE: Boschert

Size: 10.75 x 6.2 x 2.25 inches including heatsink.

Rating: in 110/220 ac (strap change) Out: 81 watts

Outputs: + 5 volts - 8.0 amps
+12 volts - 2.4 amps
+12 volts - 2.4 amps
+12 volts - 2.1 amps
-12 volts - .40 amps

Mating Connectors: Molex connectors — Load reaction: Automatic short circuit recovery

Price: $49.95
2 or more $39.95 each

ADD $ 7.50 each for S/H

Also: Dysan made Diskettes - 8" SSDD Box of 10 — $18.00
We pay Shipping in U.S.A. & Canada

DATA-COMP
5900 Cassandra Smith Rd
Hixson, TN 37343

(615)842-4600
For Ordering
Telex 5106006630

SOFTWARE FOR 680x AND MSDOS

SUPER SLEUTH DISASSEMBLERS
EACH $99-FLEX $101-OS/9 $100-UNIFLEX
OBJECT-ONLY versions: EACH $50-FLEX,OS/9,COCO
interactively generate source on disk with labels, include xref, binary editing
specify 6800,1,2,3,5,8,9/6502 version or Z80/8080,5 version
OS/9 version also processes FLEX format object file under OS/9
COCO DOS available in 6800,1,2,3,5,8,9/6502 version (not Z80/8080,5) only

CROSS-ASSEMBLERS (REAL ASSEMBLERS, NOT MACRO SETS)
EACH $50-FLEX,OS/9,UNIFLEX,MSDOS ANY 3 $100 ALL $200
specify for 180x,6502,6801,6804,6805,6809,Z8,Z80,8048,8051,8085,68000
modular, free-standing cross-assemblers in C, with load/unload utilities and macros
8-bit (not 68000) sources for additional $50 each, $100 for 3, $300 for all

DEBUGGING SIMULATORS FOR POPULAR 8-BIT MICROPROCESSORS
EACH $75-FLEX $100-OS/9 $80-UNIFLEX
OBJECT-ONLY Versions: EACH $50-COCO FLEX,COCO OS/9
interactively simulate processors, include disassembly formatting, binary editing
specify for 6800/1, (14)6805, 6502, 6809 OS/9, Z80 FLEX

ASSEMBLER CODE TRANSLATORS FOR 6502, 6800/1, 6809
6502 to 6809 — $75-FLEX $85-OS/9 $80-UNIFLEX
6800/1 to 6809 & 6809 to position-ind. — $50-FLEX $75-OS/9 $60-UNIFLEX

FULL-SCREEN XBASIC PROGRAMS with cursor control
AVAILABLE FOR FLEX, UNIFLEX, AND MSDOS

DISPLAY GENERATOR/DOCUMENTOR	$50 w/source, $25 without
MAILING LIST SYSTEM	$100 w/source, $50 without
INVENTORY WITH MRP	$100 w/source, $50 without
TABULA RASA SPREADSHEET	$100 w/source, $50 without

DISK AND XBASIC UTILITY PROGRAM LIBRARY
$50-FLEX $30-UNIFLEX/MSDOS
edit disk sectors, sort directory, maintain master catalog, do disk sorts,
resequence some or all of BASIC program, xref BASIC program, etc.
non-FLEX versions include sort and resequencer only

CMODEM TELECOMMUNICATIONS PROGRAM
$100-FLEX,OS/9,UNIFLEX,MS-DOS
OBJECT-ONLY versions: EACH $50
menu-driven with terminal mode, file transfer, MODEM7, XON-XOFF, etc.
for COCO and non-COCO; drives internal COCO modem port up to 2400 Baud

DISKETTES & SERVICES

5.25" DISKETTES
EACH 10-PACK $12.50-SSSD/SSDD/DSDD
American-made, guaranteed 100% quality, with Tyvek jackets, hub rings, and labels

ADDITIONAL SERVICES FOR THE COMPUTING COMMUNITY

CUSTOMIZED PROGRAMMING
we will customize any of the programs described in this advertisement or in our brochure for specialized customer use or to cover new processors; the charge for such customization depends upon the marketability of the modifications.

CONTRACT PROGRAMMING
we will create new programs or modify existing programs on a contract basis, a service we have provided for over twenty years; the computers on which we have performed contract programming include most popular models of mainframes, including IBM, Burroughs, Univac, Honeywell, most popular models of minicomputers, including DEC, IBM, DG, HP, AT&T, and most popular brands of microcomputers, including 6800/1, 6809, Z80, 6502, 68000, using most appropriate languages and operating systems, on systems ranging in size from large telecommunications to single board controllers; the charge for contract programming is usually by the hour or by the task.

CONSULTING
we offer a wide range of business and technical consulting services, including seminars, advice, training, and design, on any topic related to computers; the charge for consulting is normally based upon time, travel, and expenses.

Computer Systems Consultants, Inc.
1454 Latta Lane, Conyers, GA 30207
Telephone 404-483-4570 or 1717

We take orders at any time, but plan long discussions after 6, if possible.

Contact us about catalog, dealer, discounts, and services.
Most programs in source: give computer, OS, disk size.
25% off multiple purchases of same program on one order.
VISA and MASTER CARD accepted; US funds only, please.
Add GA sales tax (if in GA) and 5% shipping.
(UNI)FLEX tm Technical Systems Consultants; OS/9 Microware; COCO Tandy; MSDOS Microsoft.

SOFTWARE FOR THE HARDCORE

" FORTH PROGRAMMING TOOLS from the 68XX&X "
" FORTH specialists — get the best!! "

NOW AVAILABLE — A variety of rom and disk FORTH systems to run on and/or do TARGET COMPILATION for

6800, 6301/6801, 6809, 68000, 8080, Z80

Write or call for information on a special system to fit your requirement.

Standard systems available for these hardware—

EPSON HX-20 rom system and target compiler
6809 rom systems for SS-50, EXORCISER, STD, ETC.
COLOR COMPUTER
6800/6809 FLEX or EXORCISER disk systems.
68000 rom based systems
68000 CP/M-68K disk systems, MODEL II/12/16

tFORTH is a refined version of FORTH Interest Group standard FORTH, faster than FIG-FORTH. FORTH is both a compiler and an interpreter. It executes orders of magnitudes faster than interpretive BASIC. MORE IMPORTANT, CODE DEVELOPMENT AND TESTING is much, much faster than compiled languages such as PASCAL and C. If Software DEVELOPMENT COSTS are an important concern for you, you need FORTH!

firmFORTH™ is for the programmer who needs to squeeze the most into roms. It is a professional programmer's tool for compact rommable code for controller applications.

™ tFORTH and firmFORTH are trademarks of Talbot Microsystems
™ FLEX is a trademark of Technical Systems Consultants, Inc
™ CP/M-68K is trademark of Digital Research, Inc

tFORTH™
from TALBOT MICROSYSTEMS
NEW SYSTEMS FOR
6301/6801, 6809, and 68000

---> tFORTH SYSTEMS <---

For all FLEX systems: GIMIX, SWTP, SSB, or EXORcisor Specify 5 or 8 inch diskette, hardware type, and 6800 or 6809.

- tFORTH — extended fig FORTH (1 disk) with fig line editor. — $100 ($15)
- tFORTH + — more! (3 5" or 2 8" disks) adds screen editor, assembler, extended data types, utilities, games, and debugging aids. — $250 ($25)
- TRS-80 COLORFORTH — available from The Micro Works
- firm FORTH — 6809 only. For target compilations to rommable code. Automatically deletes unused code. Includes HOST system source and target nucleus source. No royalty on targets. Requires but does not include tFORTH +. — $350 ($10)
- FORTH PROGRAMMING AIDS — elaborate decompiler $150
- tFORTH for HX-20, in 16K roms for expansion unit or replace BASIC — $170
- tFORTH/68K for CP/M-68K 8" disk system — $290
  Makes Model 16 a super software development system.
- Nautilus Systems Cross Compiler
  — Requires: tFORTH + HOST + at least one TARGET:
  — HOST system code (6809 or 68000) — $200
  — TARGET source code: 6800-$200, 6301/6801—$200
  same plus HX-20 extensions— $300
  6809—$300, 8080/Z80—$200, 68000—$350

Manuals available separately — price in ( ).
Add $6 system for shipping, $15 for foreign air.

TALBOT MICROSYSTEMS 1927 Curtis Ave., Redondo Beach, CA 90278 (213) 376 9941

Coming Soon!

68000 products running under FLEX™

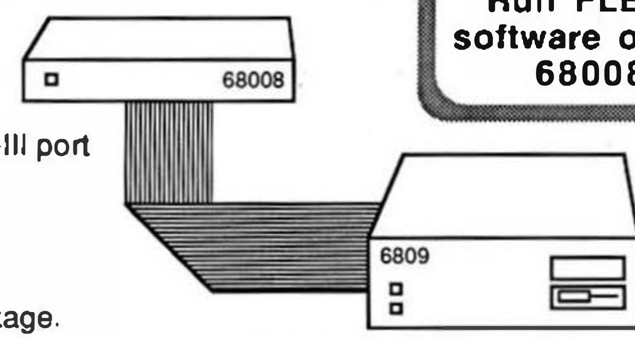
PLµS-68k (PL/9 for the 68000)

- Built-in screen editor
- Built-in source-level debugger
- Built-in assembler
- Byte, Integer, Long and Real variables
- Signed or unsigned variables
- Single-pass compiler
- Direct source to object
- Compiles over 1000 lines/min
- Requires second processor and any FLEX™ system with a PIA port

99% Compatible with PL/9

The second processor module (included with the compiler):

- 10MHz 68008 CPU
- 512K bytes RAM
- Case and power supply
- Plugs into Windrush UPROM-III port



Run FLEX™ software on the 68008

Interface software included:

- Program loader
- 68000 FLEX™ interface package.

Other Software:

68000 Assembler   68000 System Monitor
Programmer's Editor

For further information, phone or write:

Worstead Laboratories
North Walsham
Norfolk NR28 9SA
England

Tel (44) 692 404086
Telex 975548 WMICRO G

WINDRUSH
Micro Systems Ltd.

'68' MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Master Charge ☐ — VISA ☐

Card # ____________ Exp. Date ____________

For ☐ 1-Year ☐ 2 Years ☐ 3 Years

Enclosed: $ ____________

Name ____________

Stree

City ____________ State ____________ Zip ____________

My Computer Is: ____________

Subscription Rates
(Effective March 3, 1985)

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

* Foreign Surface: Add $12.00 per Year to USA Price.
* Foreign Airmail: Add $48.00 per Year to USA Price.
* Canada & Mexico: Add $ 9.50 per Year to USA Price.
* U.S. Currency Cash or Check Drawn on a USA Bank

68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

(615)842-4600

Telex 5106006630

MasterCard

LLOYD I/O INC.

Lloyd I/O is a computer engineering corporation providing software and hardware products and consulting services.

19535 NE GLISAN • PORTLAND, OR 97230 (USA)
PHONE: (503) 666-1097 • TELEX: 910 380 5448 LLOYD I O

New Product!

CRASMB™ CROSS ASSEMBLER NOW AVAILABLE FOR OS9/68000

LLOYD I/O announces the release of the CRASMB 8 Bit Macro Cross Assembler for Microware's OS9 disk operating system for the 68000 family of microprocessors. In recent increasing demand for the OS9/68000 version of CRASMB, LLOYD I/O has translated its four year old CRASMB for the OS9/6809 and FLEX/6809 to the OS9/68000 environment.

CRASMB supports assembly language software development for these microprocessors: 1802, 6502, 6800, 6801, 6303, 6804, 6805, 6809, 6811, TMS 7000, 8048/family, 8051/family, 8080/85, Z8, and the Z80. CRASMB is a full featured assembler with macro and conditional assembly facilities. It generates object code using 4 different formats: none, FLEX, Motorola S1-S9, and Intel Hex. Another format is available which outputs the source code after macro expansion, etc. CRASMB allows label (symbols) length to 30 characters and has label cross referencing options.

CRASMB for OS9/68000 is available for $432 in US funds only. It may be purchased with VISA/MASTERCHARGE cards, checks, US money orders, or US government (federal, state, etc.) purchase orders. NOTE: please add $5 shipping in the USA and use your street address for UPS shipments. Add $30 for all overseas orders. CRASMB for OS9/6809 and FLEX/6809 cost $399 plus shipping.

You may contact Frank Hoffman at LLOYD I/O, 19535 NE Glisan, Portland, Oregon, 97230. Phone: (503) 666-1097. Telex: 910 380 5448, answer back: LLOYD I O. Easylink: 62846110. See list of distributors below.

VISA, MC, COD, CHECKS, ACCEPTED

USA:	LLOYD I/O (503 666 1097),	S.E. MEDIA (800 338 6800)
England:	Vivaway (0582 423425),	Windrush (0692 405189)
Germany:	Zacher Computer (65 25 299),	Kell Software (06203 6741)
Australia:	Paris Radio Electronics (344 9111)	
Japan:	Microboards (0474) 22-1741	Seikou (03) 832-6000
Switzerland:	Elsoft AG (056 86 27 24)	
Sweden:	Micromaster Scandinavian AG (018 - 138595)	

K-BASIC, DO, SEARCH and RESCUE UTILITIES
PATCH, CRASMR, and CRASMB 16.32 are trademarks of LLOYD I/O
OS9 is a ™ of Microware, FLEX is a ™ of TSC

OS-9™ SOFTWARE

SDISK—Standard disk driver module allows the use of 35, 40, or 80 track double sided drives with COCO OS-9 plus you can read/write/format the OS-9 formats used by other OS-9 systems. **$29.95**

SDISK + BOOTFIX—As above plus boot directly from a double sided diskette **$35.95**

FILTER KIT #1—Eleven OS-9 utilities for "wild card" directory lists, copies, moves, deletes, sorts, etc. Now includes disk sector edit utility also. **$29.95** ($31.95)

FILTER KIT #2—Macgen command macro generator builds new commands by combining old ones with parameter substitution, 10 other utilities. **$29.95** ($31.95)

HACKER'S KIT #1—Disassembler and related utilities allow disassembly from memory, file. **$24.95** ($26.95)

PC-XFER UTILITIES—Utilities to read/write and format MS-DOS™ diskettes on CoCo under OS-9. Also transfer files between RS disk basic and OS-9. **$45** (version now available for SSB level II systems, inquire).

CCRD 512K RAM DISK CARTRIDGE—Requires RS Multipak Interface; with software below creates OS-9 RAM disk device. **$259**

CCRDV OS-9 Driver software for above. **$20**

BOLD prices are CoCo OS-9 format disk, other formats (in parenthesis) specify format and OS-9 level. All orders prepaid or COD, VISA and MasterCard accepted. Add $1.50 S&H on prepaid, COD actual charges added.

SS-50C

1 MEGABYTE RAM BOARD

Full megabyte of ram with disable options to suit any SS-50 6809 system. High reliability, can replace static ram for a fraction of the cost, $699 for 2 Mhz or $799 for 2.25 Mhz board assembled, tested and fully populated.

2 MEGABYTE RAM DISK BOARD

RD2 2 megabyte dedicated ram disk board for SS-50 systems. Up to 8 boards may be used in one system. $1150; OS-9 drivers and test program, $30.

(Add $6 shipping and insurance, quantity discounts available.)

D.P. Johnson, 7655 S.W. Cedarcrest St.
Portland, OR 97223 (503) 244-8152
(For best service call between 9-11 AM Pacific Time.)

OS-9 is a trademark of Microware and Motorola Inc.
MS-DOS is a trademark of Microsoft, Inc.

COMPILER EVALUATION SERVICES
By: Ron Anderson

The S.E. MEDIA Division of Computer Publishibg Inc., is offering the following SUBSCRIBER SERVICE:

COMPILER COMPARISON AND EVALUATION REPORT

Due to the constant and rapid updating and enhancement of numerous compilers, and the different utility, appeal, speed, level of communication, memory usage, etc., of different compilers, the following services are now being offered with periodic updates.

This service, with updates, will allow you who are wary or confused by the various claims of compiler vendors, an opportunity to review comparisons, comments, benchmarks, etc., concerning the many different compilers on the market, for the 6809 microcomputer. Thus the savings could far offset the small cost of this service.

Many have purchased compilers and then discovered that the particular compiler purchased either is not the most efficient for their purposes or does not contain features necessary for their application. Thus the added expense of purchasing additional compiler(s) or not being able to fully utilize the advantages of high level language compilers becomes too expensive.

The following COMPILERS are reviewed initially, more will be reviewed, compared and benchmarked as they become available to the author:

PASCAL "C" GSPL WHIMSICAL PL/9

Initial Subscription - $ 39.95
(includes 1 year updates)
Updates for 1 year - $ 14.50

S.E. MEDIA - C.P.I.
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
(615) 842-4601

68000 68020 68010

68008 6809 6800

Write or phone for catalog.

AAA Chicago Computer Center

120 Chestnut Lane — Wheeling IL 60090
(312) 459-0450

Technical Consultation available most weekdays from 4 PM to 6 PM CST

A Powerful 1 - 2 - 3 Combination

68008

68010

68000

68020

1. Stylo-Graph Word Processor
   Stylo-Merge Text Formatter
   Stylo-Spell 42,000 Word dictionary
2. Motorola 68000 Microprocessors
3. The 68K OS9 Operating System

All the Stylo programs are written in 68K assembly code making their performance second to none. The ability to always see on the screen what your printout will look like saves time and makes your work easier.

Why settle for less than the best?
Check it out today!
Call or write for catalog

Stylo Software, Inc.
PO Box 916 482 C Street
IDAHO FALLS, IDAHO 83402
(208) 529-3210

VISA OR MASTERCARD ACCEPTED

Beyond Pascal, 'C', and ADA® there is a better programming language: QPL

An easier, faster method of developing high quality programs is yours with QPL. This is a language of unmatched power and convenience which can stimulate your creative abilities.

QPL is very efficient without being terse. It's EASY TO READ & WRITE! A program written in Pascal, 'C', Basic, or Cobol can usually be written in about 70% FEWER LINES in QPL. The powerful QPL compiler manages the details of programming, allowing you to concentrate on the problem to be solved.

Many Applications:

- Business data processing
- Computer aided instruction / games
- Expert systems / artificial intelligence
- Text processing, encoding / decoding
- High precision math applications
- Systems utility programs
- Robotics
- Industrial microcomputer applications

No matter what type of programming you do, QPL has features designed to speed up development and reduce maintainance.

QPL is as easy to learn as BASIC, and more powerful than Pascal. The clearly written 90 page manual has over 30 complete example programs.

The QPL compiler and run time library is written in assembler language for maximum speed and minimum space. It includes a linking loader to minimize the final program size.

Some features of QPL are:

- Exact Arithmetic — no rounding or truncation.
  Extra wide range: (10 exp ± 32000)
- Unlimited length variable names and strings.
- Simple branch & loop methods, no nesting problems.
- Powerful string processing commands: alternation, concatenation, pattern matching, and more.
- High efficiency file formatting (about twice the space efficiency of Pascal and Basic files).
- Automatic variable sizing; no field overflows.
- Name Indirection for easy data chaining.
- Conformant arrays hold mixed data types.
- Compatible with Pascal, Basic, Cobol, Assembly.
- Simple input and output methods & printer control.
- Fast, efficient compilation.
- Symbolic tracing for fast de-bug.

Special Offer Save 50%

Language brochure with example program	FREE
Demonstration disk + mini-manual	$10.00
Full 90 page personal or business manual	$24.95
Full manual (pers. or bus.) + demo disk & binder	$34.95

Personal System: runs under Flex: $295.
Runs full language, uses smaller disk space. **Now $147**

Business System: runs under Flex: $695.
Faster operation, free run time licence. **Now $347**

Free User's Group Membership with compiler purchase.

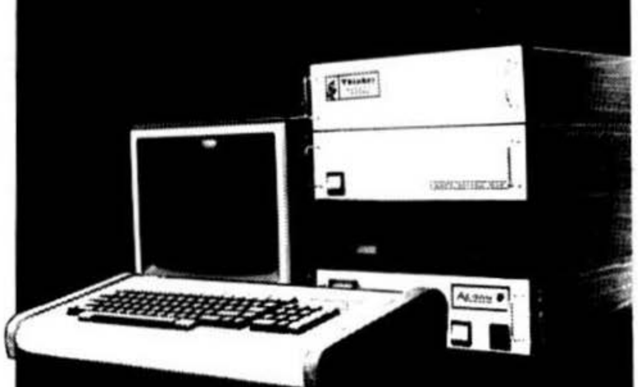
Visa & Master Card welcome.

CPU

Compiler Products Unlimited, Inc. *6712 E. Presidio, Scottsdale, AZ 86254* *(602) 991-1657*

ACORN

COMPUTER SYSTEMS SS-50C



MODULES - BARE CARDS - KITS - ASSEMBLED & TESTED

Stackable Modules	KIT	A&T
20 amp POWER SUPPLY w/fan w/Disk protect relay	350.00	400.00
DISK CABINET w/regs. & cables less DRIVES	200.00	250.00
MOTHER BOARD, 8 SS-50c, 8 SS-30c NMI button	225.00	325.00

Item	Bare	KIT	A&T
IT3 - INTERRUPT TIMER 1, 10, 100 per sec.	19.95	29.95	39.95
PB4 - INTELLIGENT PORT BUFFER Single board comput.	39.95	114.95	139.95
DPIA - Dual PIA parallel port, 4 buffered I/Os	24.95	69.95	89.95
XADR - Extended Addressing BAUD gen. PIA port	29.95	69.95	89.95
MB8 - MOTHER BOARD SS-50c w/BAUD gen.	64.95	149.95	199.95
P168 - 168K PROM DISK 21, 2764 EPROMs	39.95	79.95	109.95
FD88 - Firmware development 2, 8K blocks	39.95	84.95	114.95
XMPR - 2764 PROM burner adapt. for 2716 BURNER		19.95	------
CHERRY Keyboard w/Cabinet 96 key capacitive		249.95	------
TAXAN 12", 18 Mhz MONITOR GREEN		------	149.95
AMBER		------	159.95
4 MODULE CABINET - unfinished		150.00	------
POWER SUPPLY w/disk protect		250.00	------

Color Computer

MONOLINK - 20 Mhz Monochrome video driver	15.00	20.00
CC30 PORT BUS w/power supply 5 SS-30, 2 Cart	169.95	199.95
POWER BOX 6 switched outlets transient suppression	29.95	39.95
RS-232 3-switched ports for above	ADD +20.00	+25.00

Write for FREE Catalog
ADD $3.00 S&H PER ORDER
WIS. ADD 5% SALES TAX

ACS INC
11931 W. Bluemound Road
MILWAUKEE, WIS. 53226
(414) 257-0300

68' MICRO JOURNAL

Disk-1 Filesort, Minicat, Minicopy, Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk-2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn, **Lifetime
Disk-3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk-4 Mailing Program, *Finddat, *Change, *Testdisk.
DISK-5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk-6 **Purchase Order, Index (Disk file indx)
Disk-7 Linking Loader, Rload, Harkness
Disk-8 Crtest, Lanpher (May 82)
Disk-9 Datecopy, Diskfix9 (Aug 82)
Disk-10 Home Accounting (July 82)
Disk-11 Dissembler (June 84)
Disk-12 Modem68 (Revised June 86)
Disk-13 *Initmf68, Testmf68, *Cleanup, *Diskalign, Help, Date.Txt
Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib
Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo)
Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc
Disk-17 Match Utility, RATBAS, A Basic Preprocessor
Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray)
Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist), Dragon.C, Grep.C, LS.C, FDUMP.C
Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23 ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven Language Recognition Utility, Anderson March 1986.
Disk-24 68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25 KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26 Compacta UniBoard Review, Code & Diagram. Burlinson March 1986.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by '68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

8" Disk $ 14.95 5" Disk $ 12.95

68' Micro Journal
5900 Cassandra Smith Rd. Hixson, Tn. 37343
(615) - 842 -4600

* Indicates 6800
** Indicates BASIC SWTPC or TSC
6809 no Indicator
Foreign Orders Add $4.50 for Surface Mail
or $7.00 for Air Mail
* All Currency in U.S. Dollars

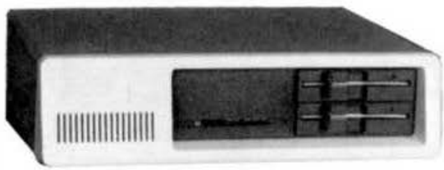
Telex 5106006630

PT-69 SINGLE BOARD COMPUTER SYSTEMS

NOW WITH WINCHESTER OR FLOPPY DISK DRIVES

The proven PT-69 Single Board Computer line is expanding! Systems now can be Winchester or floppy-based. Available also in a smaller cabinet without drives for dedicated systems with no mass storage requirements.

- 1 MHZ 6809E Processor
- Time-of-Day Clock
- 2 RS 232 Serial Ports (6850)
- 56K RAM 2K/4K EPROM
- 2 8-bit Parallel Ports (6821)
- 2797 Floppy Disk Controller



Winchester System

Custom Design Inquiries Welcome

Floppy System

• PT69XT WINCHESTER SYSTEM Includes 5 MEG Winchester Drive, 2 40-track DS/DD Drives, Parallel Printer Interface • choice of OS/9 or STAR-DOS	$1795.95
• PT69S2 FLOPPY SYSTEM Includes PT69 Board, 2 DS/DD 40-TRK 5 1/4" drives, cabinet, switching power supply, OS/9 or STAR-DOS	$895.95

• PT-69A ASSEMBLED & TESTED BOARD	$279.00
• OS/9	$200.00
• STAR-DOS	$ 50.00

PERIPHERAL TECHNOLOGY
1480 Terrell Mill Rd., Suite 870
Marietta, Georgia 30067
Telex #880584

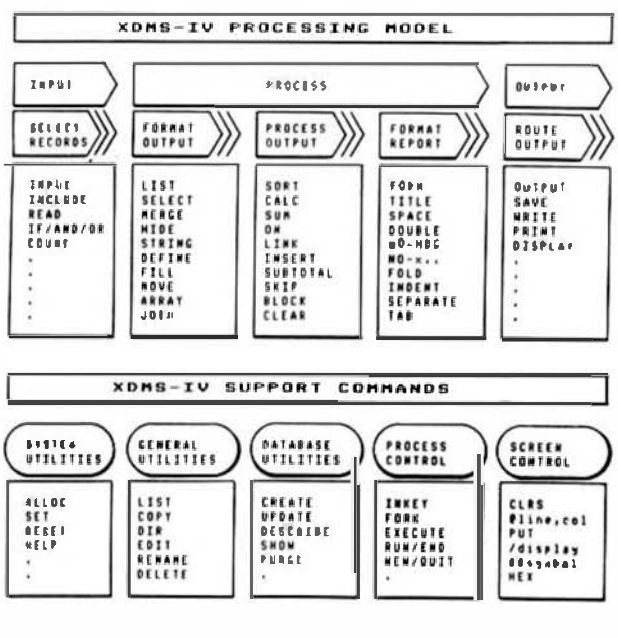
VISA/MASTERCARD/CHECK/COD

404/984-0742

CALL OR WRITE FOR ADDITIONAL CONFIGURATIONS

XDMS-IV

Data Management System



Up to 32 groups/fields per record! Up to 12 character field names! Up to 1024 byte records! Input-Process-Output (I O) command structure! Upper/Lower case commands! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriented which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV.

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting ...). The possibilities are unlimited...

XDMS-IV for 6809 FLEX, STAR-DOS, SK*DOS (5" or 8")......$350.00+P&H
Order by Phone: 615-842-4600/4601 - (VISA and MasterCard accepted)
Or write: South East Media, 5900 Cassandra Smith, Hixson, Tenn 37343

WESTCHESTER Applied Business Systems
2 Pea Pond Lane, Briarcliff Manor, N.Y. 10510 Tel 914-941-3552(Eves)
FLEX(tm) Technical Systems Consultants; SK*DOS(tm) STAR-KITS Corp.

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 5 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density DMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" DSDD Floppies, Cabinet & Power Supply $1698.88
60MB Streamer (UniFLEX-020 only) $2400.00
1.6MB Dual Speed Floppy .... (under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only) $349.67
#64 Static RAM-64K CMOS w/battery (6809 Only) $398.64
#72 256K CMOS Static RAM w/battery $998.72
#31 16 Socket PROM/ROM/RAM Board (6809 only) $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $818.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.46
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. B.P. Cable for #12 & #44 .... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS | GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 UniFLEX VM
CPU included	#05	#05	GMX III	#05	GMX III	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 intelligent	3 intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A
25MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$13.680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$17,180.20

GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE			
OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included
FLEX	Included	Included	Included
GMXBUG Monitor	Included	Included	Included
Basic 09, RunB (OS9)	Included	Included	Included
RMS (OS9)	Included	Included	Included
DO (OS9)	Included	Included	Included
VDisk for FLEX	N/A	Included	Included
RAMDisk for OS9	N/A	$125 option	Included
O-FLEX	N/A	$250 option	Included
Support ROM	N/A	N/A	Included
Hardware CRC	N/A	N/A	Included

Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

BASIC-09 and OS-9 are trademarks of Microware Systems Corp. and MOTOROLA, Inc. FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc. GIMIX, GHOST, GMX, CLASSY CHASSIS, are trademarks of GIMIX, Inc.

GIMIX Inc.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

©1985 GIMIX, INC. 11-85

Now! All the "Stars" Together!

S.E. Media

C.P.I.

Data Comp

FLEX™

STAR-DOS PLUS+™

For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Only $79.95
Reg. $250.00

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

TSC Editor
Reg $50.00
NOW $35.00

- + Read-Write-Dir RS Disk
- + Run RS Basic from Both
- + More Free Utlities
- + External Terminal Program
- + Test Disk Program
- + Disk Examine & Repair Program
- + Memory Examine Program
- + Many Many More!!!

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY, CABINET, DISK DRIVE CABLE, J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS, RS-DOS OPERATING SYSTEMS.	$469.95

* Specify What CONTROLLER You Want J&M, or RADIO SHACK

THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$125.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing......

S-50 BUS/68XX

Board and/or Computer

Terminals·CRTs·Printers

Disk Drives·etc.

REPAIRS

000422 A/E
MR. MICKEY FERGUSON
P. O. BOX 87
KINGSTON SPRINGS TN 37082
MJ

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and **confirm a Data-Comp service & shipping number!** This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

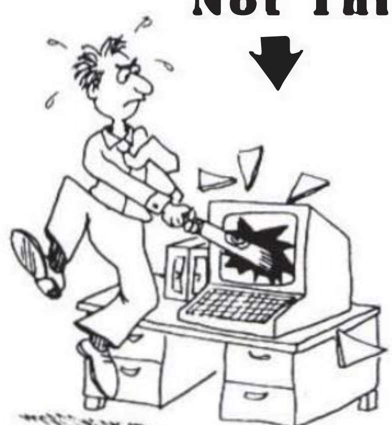
2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested, Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

This

Not This



DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4607
Telex 5106006630